SOCCRATES

SOC & CSIRT Response to Attacks & Threats
based on attack defence graphs Evaluation Systems

D4.1 AI-based Attack Detection to Detect Advanced Threats

Deliverable type:	Demonstrator
Contributing work packages:	WP4 AI-based Detection & Prediction
Due date of deliverable:	31/08/2020
Submission date:	30/10/2020 (first submission 31/08/2020, this is a re-submission to include 1st year review comments)
Dissemination level:	PU

Responsible organisation:	TNO
Editors:	Irina Chiscop
Revision:	2.0

Abstract	This deliverable provides an overview of all the algorithms and models to be used in the AI-based Attack Detection (AAD) component of the SOCCRATES platform. A set of intrusion detection tools and the over-arching reasoning engine used for correlating the produced alerts are described in detail. The efficiency of the proposed AAD component is illustrated by means of numerical experiments on the CIDDS public dataset. Integration and deployment matters are also addressed.
Keywords:	Intrusion detection, machine learning, alert fusion, alert correlation

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no. 833481
Call H2020-SU-ICT-2018 • Innovation Action • Start date: September 1st, 2019

Management summary

This deliverable provides an overview of the initial prototyping work carried out as part of Task 4.1 towards developing the AI-based Attack Detection (AAD) component of the SOCCRATES Platform. Moreover, it enriches the initial functional specification of the AAD from Deliverable 2.2 by providing in-depth explanations on the different algorithms and detection principles which will ultimately be integrated in this component.

The vision in Task 4.1 is to address a critical challenge related to intrusion detection in modern Security Operation Centers (SOC), that is dealing with the vast amount of unstructured and often noisy data from the monitored system and the high number of false alarms reported by security tools.
The solution proposed here is the integration of multiple AI-based intrusion detection tools which were developed by the SOCCRATES partners and an event-based reasoning engine, capable of combining alerts from these tools to achieve more accurate detection results. It is expected that this approach would facilitate correlation of various security events and understanding their collective meaning.

In order to demonstrate the advantage of the proposed approach, a numerical experiment was set up. Three network-based anomaly detection tools (ABC Tool, L-ADS and NIDS-AE) together with two supervised attack detection methods (DNN, DBN) were applied to a public intrusion detection dataset, the CIDDS. Their outputs were then submitted to a custom-made evidential network, that performs reasoning about the hosts and system state. The obtained results show that the integrated evidential network achieves in most cases an accuracy rate as high as the anomaly detection tools (ABC, L-ADS and AE). In particular, the precision of recognizing the normal system state was improved with the evidential network to a value of 0.989. This means that in practice, using the integrated approach may lead to a direct reduction of the false negatives. The detection of ongoing attacks in the integrated approach (evidential network) showed a slight performance improvement in comparison to the individual tools, given by a small increase of the recall score (0.997 final recall score of the evidential network).

The CIDDS dataset is quite limited in the number and complexity of attacks, therefore the improvements achieved by tools integration are not very significant. Nevertheless, the accuracy of combined solution still yields results at least as good as the best performing anomaly detection algorithm, hence, we can deduce that the evidential network is capable of handling contradictory evidence.
The next steps in task T4.1 include collecting training data from the pilot sites, thus extending the method with other data sources and anomaly detection tools to get more context and improve reasoning capabilities. Furthermore, it is intended to perform an extensive evaluation of the proposed approach within the pilot demonstration by among others the adversary emulation using tools such as Atomic Red Team covering advanced attack tactics from the MITRE ATT&CK framework.

Contributor(s)	Ewa Piatkowska (AIT) Paul Smith (AIT) Martin Teuffenbach (AIT) Agron Bajraktari (AIT) David Allison (AIT) Rubén Trapero (ATOS) Alejandro García (ATOS) Irina Chiscop (TNO) Federico Falconieri (TNO) Pieter Venemans (TNO)
Reviewers(s)	Mathias Ekstedt (KTH) (version 1) Frank Fransen (TNO) (version 2)

Security Assessment	**See deliverables table for security assessment requirements**
Approval Date	27/08/2020
Remarks	no changes required, checked and approved for publication

Version 2, October 30th 2020

Revision history

Date	**Section**	**Change**
31/08/2020	*All*	*Submission of version 1.0*
30/10/2020	*Section 1.2.1*	*Clarified the purpose and main contribution (innovation) of the Task 4.1 and this deliverable. As requested by the reviewers in the 1st year review.*
30/10/2020	*Section 3.3.3*	*Added explanation on proposed advanced attack detection with respect to MITRE ATT&CK framework. As requested by the reviewers in the 1st year review.*
30/10/2020	*Section 4.1.2*	*Additional section explaining the process of data selection and why the CIDDS dataset was particularly chosen for the proof of concept. As requested by the reviewers in the 1st year review.*
30/10/2020	*Section 4.4.1*	*Added more details on planned evaluation and next steps in Task 4.1. As requested by the reviewers in the 1st year review.*

30/10/2020	*Section 5.1*	*Explained how the detection tools and the AAD will be trained and assessed at the pilot sites. As requested by the reviewers in the 1st year review.*
30/10/2020	*Section 6*	*Summary of evaluation plans with respect to pilot sites: data collection for training and testing. As requested by the reviewers in the 1st year review.*
30/10/2020	*All*	*Submission of version 2.0*

TABLE OF CONTENTS

1 INTRODUCTION 6
1.1 The SOCCRATES project 6
1.2 This deliverable 7
1.3 Structure of this deliverable 8
2 AI-BASED DETECTION TOOLS 9
2.1 Overview 9
2.2 DNS Ninja 10
2.3 ABC Tool 13
2.4 L-ADS 16
2.5 A-Miner 19
2.6 Deep Learning Algorithms for Network Intrusion Detection 21
2.7 Autoencoder 23
3 EVENT-BASED REASONING 25
3.1 REASENS Framework 26
3.2 Reasoning with Evidential Networks 27
3.3 SOCCRATES tools integration 31
4 PROOF OF CONCEPT 36
4.1 Experimental setup 36
4.2 CIDDS dataset 38
4.3 Anomaly and Attack Detection Results 40
4.4 Evidential Network Results 44
5 DEPLOYMENT PLANS 47
5.1 Overview 47
5.2 Deployment options 48
5.3 Pilot Integration (Phases) 50
6 CONCLUSIONS 52
7 REFERENCES 53
8 ABBREVIATIONS 54

1 Introduction

This section introduces the SOCCRATES project and defines the goals of this deliverable.

1.1 The SOCCRATES project

SOCCRATES (SOC & CSIRT Response to Attacks & Threats based on attack defence graphs Evaluation Systems) is an EU funded project under the Horizon2020 programme that has the following main challenge:

> ***How can SOC and CSIRT operations effectively improve their capability in detecting and managing response to complex cyber-attacks and emerging threats, in complex and continuously evolving ICT infrastructures while there is a shortage of qualified cybersecurity talent?***

The main objective of SOCCRATES is to develop and implement a security automation and decision support platform (**'the SOCCRATES platform')** that will significantly improve an organisation's capability (usually implemented by a SOC and/or CSIRT) to quickly and effectively detect and respond to new cyber threats and ongoing attacks.

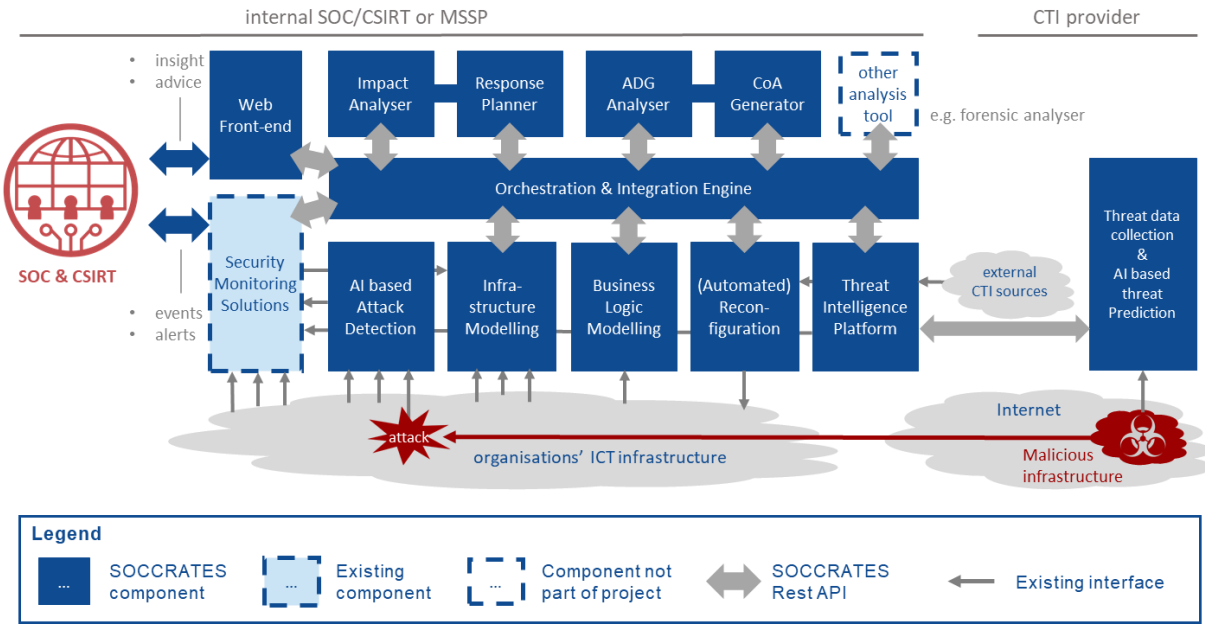

Figure 1.1 – The SOCCRATES platform

The SOCCRATES platform (see **Figure 1.1**) consists of an orchestrating function and a set of innovative components for automated infrastructure modelling, attack detection, cyber threat intelligence utilization, threat trend prediction, and automated analysis using attack defence graphs and business impact modelling to aid human analysis and decision making on response actions, and enable the execution of defensive actions at machine-speed.

SOCCRATES has the following concrete project objectives:

1. Deliver the SOCCRATES platform consisting of an orchestration function and a unique integration of innovative background solutions that seamlessly work together.
2. Show that the SOCCRATES platform can improve SOC operations by evaluating the SOCCRATES platform in two diverse real-life pilot environments.
3. Examine and illustrate the benefits of automation for selected SOC activities to help manage the cyber security skills gap in organizations.
4. Prepare for successful exploitation by the SOCCRATES partners of the individual innovated components and the integrated SOCCRATES platform in commercial products that are offered to the market and are available for the European (business) community.

Please visit www.soccrates.eu for more information on the SOCCRATES project.

1.2 This deliverable

1.2.1 Purpose and Scope

Deliverable 4.1 (D4.1) is part of Task 4.1 "Advanced Attack Detection using Artificial Intelligence" and describes the current progress towards developing the *AI-based Attack Detection (AAD)* component of the SOCCRATES Platform. The goal of this component is to integrate both existing and new machine learning algorithms that enable fast and accurate detection of cyber-security incidents. This document provides an overview of all the modelling and algorithmic approaches which will be part of the aforementioned AAD component and outlines further deployment plans. There is no further development or innovation planned for the individual detection tools because each of them has been in use at the partner's premises for a long time. Therefore, their quality has already been proven in practice. The main objective of the AAD is thus to improve detection capabilities using an integrated solution to prioritise events.

1.2.2 Approach

The SOCRATES AAD component aims to detect advanced threats and support the role of SOC analysts, who currently need to monitor and inspect an extremely large volume of events. Moreover, techniques relying on signature-based detection tools and rule-based correlation in the SIEM are no longer sufficient to keep up with the growing complexity of cyberattacks. In Task 4.1, a solution is proposed to tackle these challenges by combining alerts (events) from different AI-based detection tools to perform event-based reasoning about potential root causes of those alerts. The functional architecture of the AAD Component is shown in Figure 1.2. As described in Deliverable 2.2 (Tesink, 2020), the functionalities provided by the AAD component include host-based and network-based anomaly detection on a variety of data sources, event-based reasoning applied to collected security events to assess system state with higher confidence. The AAD component reports anomalies found in host behaviour, network traffic and DNS queries. It is intended to be flexible and easy to interface with the existing security monitoring solutions (e.g. SIEM).
These functionalities will be demonstrated in two phases at the SOCCRATES Pilot Sites: Vattenfall 's Security Operation Center in Poland, and at mnemonic's proprietary platform for managed detection and response (Argus) located in Norway.

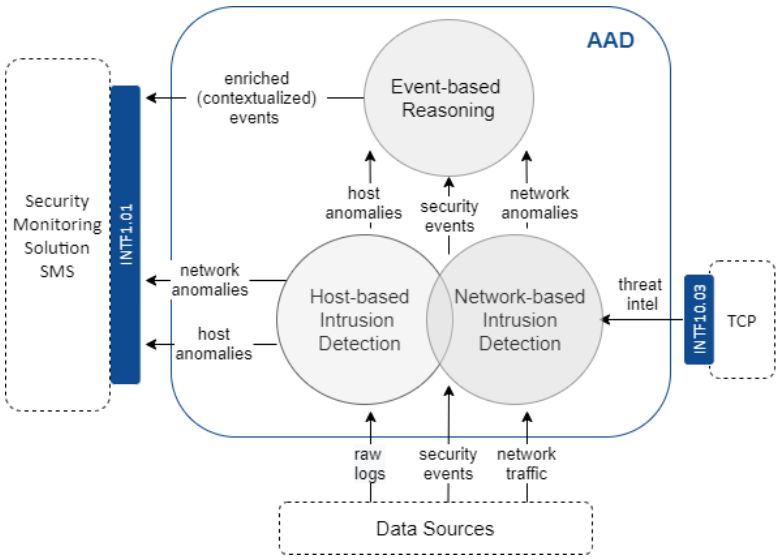

Figure 1.2: Functional Architecture of AAD component

The proposed approach is considered to have several advantages:

- Firstly, the use of state-of-the-art machine learning algorithms for anomaly detection (instead of outdated signature-based methods) increases the capability of detecting new cyber-attacks.
- Secondly, the detection algorithms use different data sources, such as NetFlow data, DNS queries, logs, etc. Therefore, a larger amount of information about the underlying infrastructure becomes available, can be monitored and utilised for enriched attack representation.
- Furthermore, integration of the tools results in normalisation of their outputs into a common (pre-defined event format), thus it becomes easier to explain and prioritise alerts.
- Finally, event-based reasoning can be achieved using established correlation and inference algorithms (Bayesian or Evidential networks), as well as modern techniques such as recurrent neural networks (RNN).

Even though event-based reasoning (or alert correlation) is usually attributed to functionalities offered at SIEM, the AAD component is not intended to replace SIEM. It offers a flexible way of integrating more advanced AI-based solutions to complement the rule-based alert correlation performed at SIEM.

The SOCCRATES partners involved in this task, namely TNO, AIT and ATOS, have extensive expertise in the field of network- and host-based intrusion detection. At the start of the project, multiple intrusion detection algorithms that were developed by the partners were available for tests and deployment, e.g. DNS Ninja, ABC Tool (TNO), L-ADS (ATOS) and AMiner (AIT). The efforts in Task 4.1 have been mainly focused on integration of these algorithms and applying suitable method for event-based reasoning to better understand the underlying system state.

1.3 Structure of this deliverable

This deliverable is organised in the following way: Section 2 provides an overview of intrusion detection algorithms developed by different SOCCRATES partners. Section 3 describes in detail the structure and components of the reasoning approach. In Section 4, a prototype of the event-based reasoning based on combined output from network-based detection tools is demonstrated by numerical experiments on a public dataset. Section 5 outlines the current deployment plans within the larger context of the SOCCRATES platform and the pilot sites. Finally, Section 6 summarizes the conclusions that can be drawn from the progress achieved until this point in time and outlines the future work.

2 AI-based Detection Tools

2.1 Overview

This section provides a detailed description of the current AI-based detection models developed by the SOCCRATES partners. The underlying models, methods and algorithms of each detection tool are addressed here, in order to gain insights into how different AI methods enhance the detection capabilities. Integration-related aspects with other components in the SOCCRATES platform are outside the scope of this documents and are thoroughly addressed in Deliverable 2.2 (Tesink, 2020).

AI-based detection techniques and algorithms can be classified in three categories:

1) Supervised learning approaches require using labels, or in other words, prior knowledge over the input samples. The goal here is to learn a function that correctly approximates the relationship between the input data and the provided labels. Supervised learning is either used for classification (for categorical data) or regression (on numerical data).
2) Unsupervised learning approaches on the other hand attempt to infer the inherent structure of the data without using explicitly provided labels. Most common examples of unsupervised learning are clustering and pattern discovery through data mining.
3) Semi-supervised learning refers to combining the two categories above in the sense that during learning, it uses a small set of labelled data together with a large number of unlabelled samples.

A categorization of AI-based Detection Tools discussed in this chapter according to the different methods of learning presented above is shown in Table 2.1.

Table 2.1: Classification of detection tools from SOCCRATES partners

Detection Tool	Type of learning	Learning method
DNS Ninja (TNO)	Supervised Unsupervised	SVM Statistical analysis
ABC Tool (TNO)	Semi-supervised	Clustering
L-ADS (ATOS)	Semi-supervised	Autoencoder
A-Miner (AIT)	Semi-supervised	Clustering, Statistical analysis
Deep Learning Algorithms for Network Intrusion Detection (AIT)	Supervised Supervised Semi-supervised	Deep Neural Network Deep Belief Network Autoencoder
Autoencoder (TNO)	Unsupervised	Autoencoder

2.2 DNS Ninja

2.2.1 Brief overview

Contributor	TNO
Model/Tool	DNS Ninja
Data input	DNS queries & responses
Data description	This tool can use as input Full DNS query and/or response messages in Ethernet frames and/or Log data DNS Server and/or Data from HTTP Proxy.
Output	Hosts with deviating statistical properties of DNS traffic, flagged DNS queries
Underlying method	For domain generation algorithms (DGA) detection: N-gram frequency, lists, dictionaries For Malware detection: Statistical properties Analysis, behaviour over time analysis For malicious host detection: Statistical deviation from the rest
Type of learning	Supervised (DGA detection) and unsupervised (host detection)
Explanation	An SVM (Support Vector Machine) is used for detecting domains created by generation algorithms and user interpretation of the statistical deviation of DNS queries is used to find anomalous hosts.
Labels	Required for DGA detection Not required for host detection
Event indicator	Predicted class from SVM for DGA detection User-defined threshold over slope in scatter plot for host detection
Detection principle	A host is marked anomalous if its computed anomaly score exceeds the user-specified threshold.
Targeted attacks	Botnets, C&C traffic, ransomware, key loggers, data exfiltration
Reference	No formal documentation exists.

2.2.2 Purpose

DNS Ninja is a system for efficiently monitoring DNS traffic in a corporate network environment with the purpose of detecting malware and other security threats at the earliest possible stage. DNS Ninja applies Anomaly Detection, which means that no "signatures" are required, and zero-day malware and other zero-day breaches may be detected. The system uses advanced mathematical techniques that model the "normal DNS behaviour" of internal systems (laptops, servers, smartphones, etc.). Everything which differs significantly from normal is detected and reported as an anomaly. DNS Ninja is not a replacement of existing solutions (such as antivirus, DNS blacklist, IDS / IPS, next generation firewall), but an additional line of defence. DNS Ninja can be used in combination with existing SIEM (Security Information and Event Management) technology. It essentially works as a smart data pre-processor that monitors large amounts of DNS traffic in real time and reports interesting cases to the SIEM. The SIEM, in turn, can correlate these alerts with other information and trigger follow-up actions. The web interface makes it also possible to use DNS Ninja without a SIEM.

2.2.3 Data pre-processing

DNS Ninja uses as input data the DNS queries from internal workstations. In terms of pre-processing the main step that is required is linking the internal IP addresses to host names.

The incoming stream of queries and responses is examined to find mappings from internal IP addresses to host names, either from forward or reverse lookups. All mappings that are found are stored in a cache with an expiry time of 5 minutes. If an internal IP address is encountered for which no host name is present in the cache, the server performs a reverse DNS lookup by itself. This way, an almost complete mapping of internal IP addresses to local host names is obtained, which allows the users to follow the behaviour of individual hosts, even if their IP addresses change during the day. This mapping is also possible when using logs produced by the DHCP server. That is because the DHCP server and DNS resolver are set up in such a way that a reverse DNS lookup of a DHCP-assigned IP address returns a unique identification of the local machine.

The DNS data can also be used to create a list of 1000 most frequently requested external domain names (Google, Facebook, Windows update, etc.). After manual curation, this list may be used as whitelist. The full top-1000 list covers approximately 93% of all DNS queries recorded on a given workday. This list is used into one of the DNS Ninja detectors.
Another data pre-processing which is needed in one of the detectors for DGAs (domain generation algorithms) is the extraction of n-gram frequencies from the domain names for several languages.

2.2.4 Method

DNS Ninja comprises different detectors which enable the user to detect domains that may be automatically generated, identify local hosts whose DNS traffic statistics deviate too much from the general behaviour, and discover computers which have a certain periodicity of their DNS queries. Each detector relies on a different algorithm, which we separately discuss below.

Detecting DGAs

One of the components of DNS Ninja focuses on identifying DGA domains based on the properties of the domain names. This includes analysing frequencies of sequences of certain letters or symbols of fixed lengths, which are known as n-grams. In every language there are certain n-grams that occur very often and if a domain name includes a sequence of letters that is highly uncommon, that can be an indication of that domain being (randomly) generated by an algorithm and used for malicious purposes.
In DNS Ninja features such as the frequency of different n-grams are calculated using the domain name string and are used as input for a support vector machine (SVM), which is a supervised machine learning technique. This SVM is trained on labelled domains from the whitelist generated during the preprocessing, but also on well-known DGA families such as Cryptolocker, Game-Over-Zeus etc. The output of this detector is an anomaly score inferred from the loss function value of the classifier.

Detecting anomalous hosts

DNS traffic can be very useful for inspecting the activity of a certain local host. DNS queries can reveal for instance the existence of Botnets or malware attempting to connect with command-and-control (C&C) servers. Therefore, detecting hosts, i.e. workstations in the internal network, which behave differently than the large majority, is another detection purpose of DNS Ninja. The monitored DNS queries and responses are filtered into different categories based on the query type. Examples of query types may include queries from local/internal domains, queries from a whitelisted IP, or reverse lookups. In total, DNS Ninja has 24 such categories. Using different filter combinations, the detector

tries to infer a relationship between the mean and standard deviation of the number of selected queries for a particular host over a period of fixed time length (the last 4 hours for instance). If a data sample (local host) falls outside the acceptable pre-specified threshold value, then it is deemed anomalous.
The same detection principle can be applied to detect periodic DNS queries. This periodicity is relevant because certain type of malware regularly tries to connect to C&C servers by requesting DGA domains. Therefore, anomalies found with this detector could potentially mean that there are hosts infected with malware in the system.

2.2.5 Output

DNS Ninja is equipped with a Web interface which enables the user to gain a better insight into the reported anomalies through different means of visualization. The user can specify the time range, dataset and aggregation interval, as well as different filtering options (e.g. to list only queries from whitelisted IP addresses) and adjust the anomaly thresholds. Moreover, DNS Ninja also supports integration with the SIEM. The alerts are currently reported as syslog messages and can be optionally formatted as Common Event Format (CEF) messages for consumption by SIEMs such as HP ArcSight. The format of alerts can be easily modified to whatever the requirements of the SIEM solution at hand.

The anomalies detected are mainly visualized by means of graph and scatter plots. Figure 2.1 shows an example of such plot, in which all blue dots mark individual hosts. The red region marks the 'normal' data, whilst the red line represents the slope threshold specified by the user. Dots outside the shaded area will generate anomaly events when running real-time. When selecting such an anomaly point, we can inspect the DNS traffic of that particular host, by selecting a time interval and the type of query, as shown in Figure 2.2.

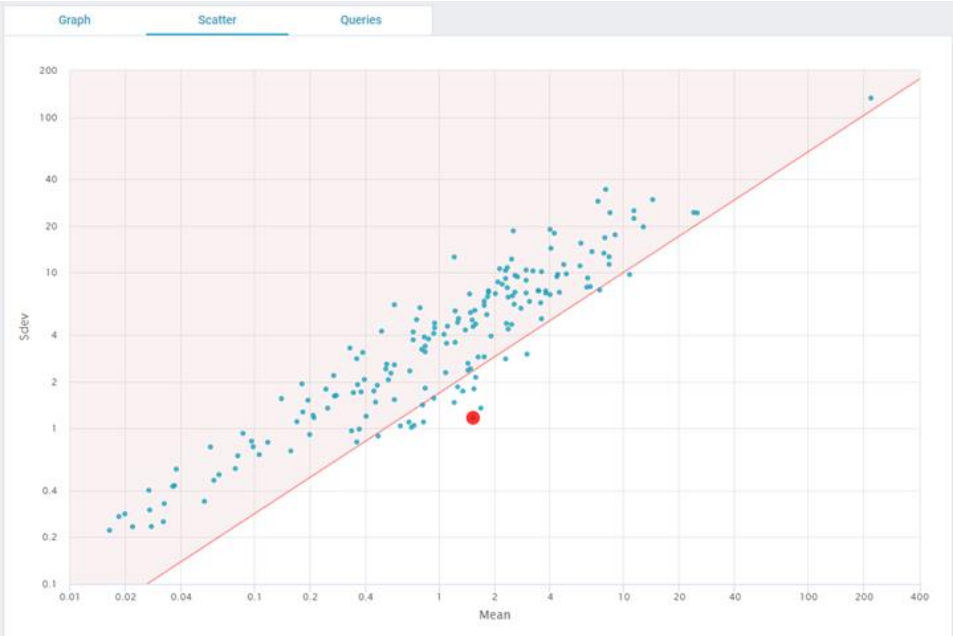

Figure 2.1: DNS Ninja scatter plot

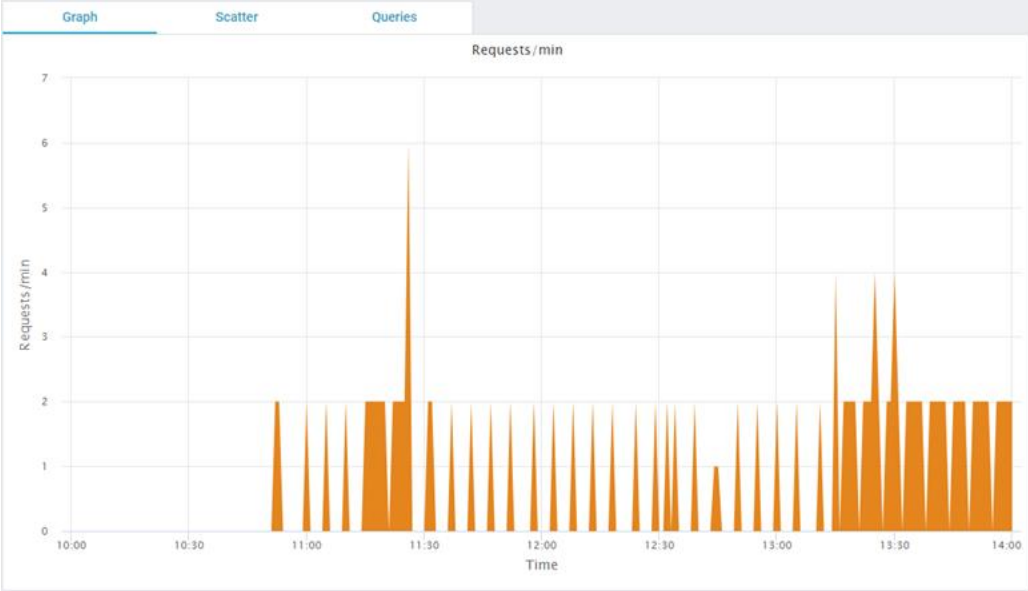

Figure 2.2: Graph plot of DNS queries of a single host

2.3 ABC Tool

2.3.1 Brief overview

Contributor	TNO
Model/Tool	ABC-tool (**A**nomaly **B**ased **C**lustering tool)
Data input	Internal Net-flow data
Data description	The tool can be used for both unidirectional/bidirectional raw flow data with the following features: flow duration, no. of bytes, no. of packets etc.
Output	A list of internal hosts which are labelled as anomalous
Underlying method	In the training phase, all individual host in the network (unique IPs) are clustered according to their behaviour. In the testing phase, anomalies are defined as: communicating with a cluster to which there was no traffic flow before, the appearance of a new cluster, increased communication to a specific cluster etc. The algorithm therefore outputs the hosts that have changed the way they behave with respect to their cluster.
Type of learning	Semi-supervised
Explanation	Three types of anomalies are defined based on the interpretation of data clusters.
Labels	Not required
Event indicator	Anomaly score
Detection principle	A sample is marked anomalous if its computed anomaly score exceeds the user-specified threshold.
Targeted attacks	Lateral movement
Reference	(Beukema, 2016)

2.3.1 Purpose

The ABC Tool was designed to perform anomaly detection in an internal network setting. Specifically, the method looks at network flows between hosts that reside in the same network. As such, the domain of attacks that can be detected is reduced. Concretely, anything that happens outside a firewall

is out of scope, while attacks happening within the confines of the firewall, like lateral movement, internal port scanning, internal C&C, and data exfiltration, are potential targets for detection.
The purpose of the method is to reduce the number of hosts that need to be scrutinized when a system or network is suspected to be under attack. It is not aimed at detecting all malicious flows with a high true positive rate. As such, it should be seen as an addition to current defence systems, but not as a replacement for an IDS. Furthermore, the use of modularity optimization yields a network of communities. These communities can provide insightful and novel perspectives on a system or network which additionally grants context to the anomalies detected. Hence another goal is to make anomalies more explainable within the context of their network.

2.3.2 Data pre-processing

The ABC Tool is aimed at modelling internal communication as a set of weighted edges among the nodes, represented by the hosts and the cluster of hosts. It takes as input data in form of source-destination communication pairs with at least two features specified, namely source and destination IDs. Optionally, other fields, such as time and weight, could be used. Examples of suitable data sources include internal DNS logs, NetFlow and file access logs. In the case of NetFlow data, we need to filter out the flows having as source or destination an IP address outside of the internal network. This step is necessary in order to focus only on the internal traffic.

2.3.3 Method

The method can be divided into two phases. The first phase aims at creating a baseline model of each community in the internal network by dividing it into communities. The second phase then uses this model to evaluate the behaviour of hosts and determines whether it is anomalous with respect to the baseline.

Creating a baseline

We start by building a directed weighted graph $G = (V, E)$ where V represents all the hosts in the network, and $e = (v_i, v_j) \in E$ if and only if v_i has a flow to v_j for $v_i, v_j \in V$. The weight w_e of an edge e is equal to the total amount of bytes sent from host v_i to host v_j. We then partition the graph into communities through modularity optimization. We achieve this by using the Louvain algorithm. This yields a list of N communities $\mathcal{C} = \{C_1, C_2, \dots, C_n\}$ where $\forall\, C_i \in \mathcal{C}, C_i \subseteq V$.
Once all communities have been formed, we know to which community each host belongs. For each inter-community connection, we construct a model which is based on the volume of traffic going from the hosts within the community to hosts from the other communities.

This process goes as follows. Assume we have arbitrary communities C_i and C_j. We take all the hosts $h \in C_i$. For all h, we calculate the total amount of bytes B_h sent to C_j. Finally, we fit all B_h to a normal distribution with parameters μ_{ij} and σ_{ij}, which will be our model for the communication from C_i to C_j. Additionally, to μ_{ij} and σ_{ij}, each host in C_i gets a scaling parameter s_h to account for consistently high (or low) amounts of traffic for that host.

Detecting anomalies

Now that we have a baseline model for every inter-community communication, we can start inspecting novel data for anomalies. To do so, we compare the network traffic of each host to their respective community models, and label it as an anomaly if we detect one of the following three deviations:

1. Increase in inter-community communication.
2. Introduction of a new community-to-community communication.
3. A community communicates to a host that was unseen before.

The first type of anomaly occurs when the communication from one host to another exceeds a certain threshold. Concretely, it should hold that $\alpha_{ab} > T$, where T is some selected threshold and

$$\alpha_{ab} = \frac{x_{ab} \cdot s_a - \mu_{ij}}{\sigma_{ij}}, \quad 1 \leq i, j \leq N,$$

where α_{ab} is the anomaly score for traffic x_{ab} from host $a \in C_i$ to host $b \in C_j$, s_a is the scaling parameter for host a, and μ_{ij} and σ_{ij} are the mean and standard deviation of the traffic from hosts in community $C_i$to hosts in community C_j. The goal of this detection is to test how well the amount of traffic fits the normal distribution of the inter-community model, taking into account the scaling parameter for that host.

The second anomaly covers the situation where a host from community C_i communicates to a host from community C_j, when the inter-community baseline model from C_i to C_j does not exist (i.e. there is no μ_{ij} and σ_{ij}). Or in other words, when there was never a host from C_i in the baseline model that communicated to a host in C_j.

The third and final anomaly occurs when a host communicates to a host that was not known before when creating the inter-community baseline models.

2.3.4 Output

The main output of the ABC Tool is a file which provides an overview of all unique IP addresses together with anomaly-related information. The anomaly list, ordered by decreasing measure of anomaly, the host, the number of anomalies, the number of new cluster-connections, number of connections to new hosts, number of irregular connections, list of tuples consisting of anomaly score (higher is more anomalous) and cluster ID (cluster that the host anomalously connects to), list of cluster ID that are anomalous for this host, list of cluster IDs for new cluster-connections, list of cluster IDs for connections to new hosts, list of cluster IDs for irregular connections.
Other output files that can be used for the purpose of anomaly inspection are the visualizations of the cluster structure for the training and test data, and the zoomed-in cluster model for each cluster. Such an example in shown in Figure 2.3. On the left side of Figure 2.3 the model for cluster 0 is displayed. On the right, the anomalous behaviour of a host in cluster 0 is shown. The red node indicates an anomalous amount of communication with cluster 1, the pink node 2 indicates new cluster communications from the host in node 0 that never happened in the training data, the light-green node 3 indicates that the host in cluster 0 communicated with new hosts that were not seen in the training data.

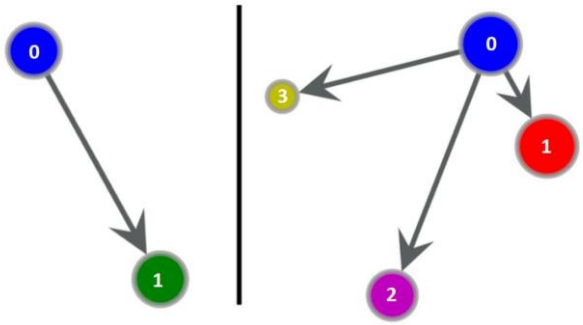

Figure 2.3: Training and testing clustering model for cluster 0

2.4 L-ADS

2.4.1 Brief overview

Contributor	Atos
Model/Tool	L-ADS (**L**ive **A**nomaly **D**etection **S**ystem)
Data input	Internal Net-flow data
Data description	The tool can be used for both unidirectional/bidirectional raw flow data with the following features: flow duration, no. of bytes, no. of packets etc.
Output	A list of internal hosts which are labelled as anomalous
Underlying method	In the training phase, the algorithm tries to learn about the flows received as input. For that reason, it is important the used flows to train must be "normal" connections. In the testing phase, it is evaluated the new flows comparing if these flows are similar to the training flows. If the difference is high the algorithm will classify as anomalous and legit in the opposite case.
Type of learning	Semi-supervised
Labels	Not required
Event indicator	Anomaly score
Detection principle	A sample is marked anomalous if its computed anomaly score exceeds the user-specified threshold.
Targeted attacks	Different attacks such as Port scan, Ping scan, Brute Force or Denial of Service.
Reference	(Gonzalez-Granadillo, 2019)

2.4.2 Purpose

The asset **L-ADS (Live Anomaly Detection System)** is developed with the aim to try to reduce the number of possible attacks in a certain internal network. This asset monitors the network flows provided by the tool NetFlow and classifies the flows as legit or as anomalous.

2.4.3 Data pre-processing

The L-ADS detects anomalies based on the flow information extracted from network traffic, in NetFlow format. NetFlow is a protocol developed by Cisco, which allows to capture different features of the network such as source IP, source port, destination IP, destination port, duration, protocols, packets, bytes, etc.

The tool used for the implementation of NetFlow is Softflowd. It supports data exports of traffic flows (between the source IP and destination IP) in different versions 1, 2 and 9 of NetFlow. The flow shall be understood as the communication between two IPs, one source IP and one destination IP.
Figure 2.4 shows an example of what is the structure generated by Softflowd.

Date first seen	Duration	Proto	Src IP Addr	Src Pt	Dst IP Addr	Dst Pt	Packets	Bytes	Flows	Flags	Tos
2017-04-12 00:02:16.977	0.000	TCP	192.168.220.11	34079	192.168.100.5	445.0	1	66	1	.A....	0
2017-04-12 00:02:20.290	0.000	TCP	EXT_SERVER	8082	192.168.200.9	53089.0	1	200	1	.AP...	32
2017-04-12 00:02:20.272	0.224	TCP	192.168.200.9	53089	EXT_SERVER	8082.0	2	319	1	.AP...	0
2017-04-12 00:02:20.901	0.003	TCP	192.168.220.7	50015	192.168.100.5	445.0	2	174	1	.AP...	0
2017-04-12 00:02:20.902	0.000	TCP	192.168.100.5	445	192.168.220.7	50015.0	1	108	1	.AP...	0
2017-04-12 00:02:24.876	0.500	TCP	192.168.100.5	445	192.168.220.16	33572.0	2	178	1	.AP...	0
2017-04-12 00:02:24.878	0.498	TCP	192.168.220.16	33572	192.168.100.5	445.0	3	240	1	.AP...	0

Figure 2.4: Softflowd dataset

The principal features of NetFlow are:

- ***Date first seen***. The start time of the flow.
- ***Duration.*** Duration of the flow.
- ***Proto.*** Protocol used in the flow.
- ***Src IP Addr.*** Source IP address.
- ***Src Pt.*** Source Port.
- ***Dst IP Addr.*** Destination IP address.
- ***Dst Pt.*** Destination Port.
- ***Flags.*** TCP flags of the flow.
- ***Bytes***. The number of bytes in the flow.
- ***Flows.*** The number of flows.
- ***Packets.*** The number of packets in the flow.
- ***Tos.*** Type of service.

In addition, the features presented below are not unique. The L-ADS try to improve the performance of the classification using some new features created from the previous features.

- ***Packets_speed.*** The number of Packets divided by the Duration.
- ***Bytes_speed.*** The number of Bytes divided by the Duration.
- ***Packets_per_flow.*** The number of Packets divided by the number of flows
- ***Bytes_per_flow.*** The number of Bytes divided by the number of flows.

These features can take numeric or categorical values. In other words, there are some features, e.g., Proto and Flags, which need to be transformed into numeric features. The categorical features are transform using dummy variables.
The final step of the feature pre-processing is the standardisation of the whole dataset (features take values from the same range).

2.4.4 Method

The L-ADS was developed in two different versions, each of which differs according to the importance of the IPs. The selected version will be chosen according to the performance obtained in the testing phase.
The first version (L-ADS version 1) models IP addresses separately. The first step performs filtering the Softflowd dataset using a certain IP. The resulting dataset only contains the flows where given IP is a source or a destination address. In this process, the original dataset generates k subsets, one for each

IP. These datasets add a new dummy column called ***is_source*** if the IP appears as source or as destination. As shown in Figure 2.5, resulting (generated) subset of the dataset in Figure 2.4 after filtering using the IP *192.168.100.5*.

Date first seen	Duration	Proto	Src IP Addr	Src Pt	Dst IP Addr	Dst Pt	Packets	Bytes	Flows	Flags	Tos	is_source
2017-04-12 00:02:16.977	0.000	TCP	192.168.220.11	34079	192.168.100.5	445.0	1	66	1	.A....	0	0
2017-04-12 00:02:20.901	0.003	TCP	192.168.220.7	50015	192.168.100.5	445.0	2	174	1	.AP...	0	0
2017-04-12 00:02:20.902	0.000	TCP	192.168.100.5	445	192.168.220.7	50015.0	1	108	1	.AP...	0	1
2017-04-12 00:02:24.876	0.500	TCP	192.168.100.5	445	192.168.220.16	33572.0	2	178	1	.AP...	0	1
2017-04-12 00:02:24.878	0.498	TCP	192.168.220.16	33572	192.168.100.5	445.0	3	240	1	.AP...	0	0

Figure 2.5: Filtered dataset by the IP 192.168.100.5

This method attempts to reduce the noise of other flows that are not relevant. In addition, we will train anomaly detection algorithms independently for all k subsets. However, potential drawback of this approach is that for certain IPs the resulting subset could have insufficient flows to train the associated algorithm which leads to a bad performance.

On the other hand, the **L-ADS version 2** uses the whole Softflowd dataset without filters. The algorithm tries to get the correlation between all the features except of features related to IP addresses (source and destination). This method only requires one model to be trained, therefore it is possible to have some noisy flows. The rest of the L-ADS age model procedure is the equivalent for the two variants. The L-ADS uses Autoencoder to get the principal characteristics of the dataset and model normal flows. It will be trained using benign traffic samples. Figure 2.6 shows the architecture of the Autoencoder.

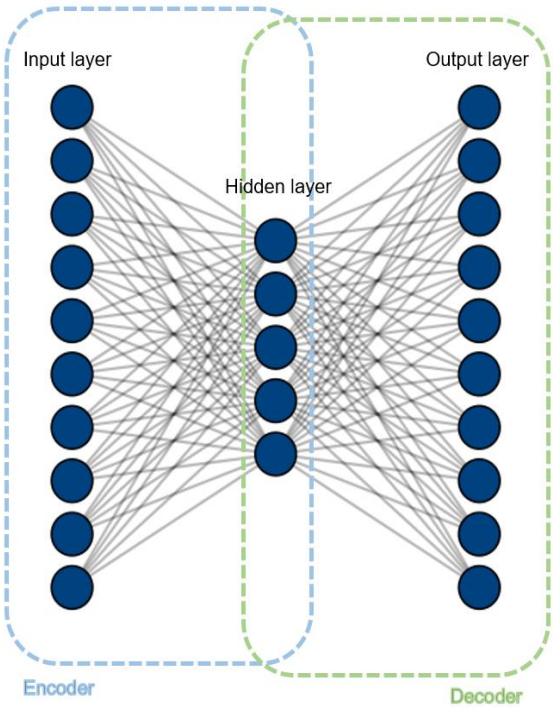

Figure 2.6: Autoencoder architecture

The Autoencoder comprises of two unique parts:

- Encoder, which receives the dataset as input and compresses the information. The output has a dimension smaller than the input.
- Decoder, which receives as input the encoder's output and decompresses the information. The output will have the same dimension as the input dataset.

The Autoencoder will be trained using the dataset as input and output so that it learns to compress and decompress the normal traffic. When the autoencoder evaluates an anomalous connection, the transformed connection should be quite different. For that, it is necessary to use a function capable of quantifying the difference between original and reconstructed sample. To this end, a ***mean square error (MSE)*** is used*:*

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - Y_i')^2,$$

where n is the number of features, Y_i Is the value of the i-th feature in the connection and Y_i' Is the value of the i-th feature in the reconstructed connection.

This function will generate a numerical value associated with the similarity of original and reconstructed flows. If the resulting value is bigger than a specific value (threshold), the connection will be categorized as anomalous. Otherwise, if the value is less than the threshold it will be categorized as normal connection. The threshold must be predefined and is derived in the process of a semi-supervised learning (i.e. threshold is adjusted using the labels of a supervised training set).

2.5 A-Miner

2.5.1 Brief overview

Contributor	AIT
Model/Tool	AMiner (logdata-anomaly-miner - lightweight tool for log checking, log analysis)
Data input	log data streams
Data description	log output from the network layer (e.g., firewalls, switches, routers) and application layer (e.g., Web servers, DNS, application servers etc.)
Output	Suspicious log entries (detected anomalies) for further analysis
Underlying method	Analysis methods include: - static check patterns like logcheck but with extended syntax and options - detection of new data elements (IPs, usernames, MAC addresses) - statistical anomalies in log line values and frequencies
Type of learning	Semi-supervised
Explanation	Types of anomalies (depending on detector modules), e.g. new path detected (service or host), variable values out of predefined ranges, unusual event frequency
Labels	Not required (model built on benign input data)
Event indicator	Unknown path in parser model; rule violation (whitelisted approach)
Detection principle	Log line cannot be parsed with parser model, and/or log lines do not follow certain predefined rules
Targeted attacks	Network traffic anomalies
Reference	(Wurzenberger M. L., 2019)

2.5.2 Purpose

AMiner (logdata **A**nomaly **Miner**) is part of the ÆCID system, a log-based anomaly detection system operating on logs collected from the network layer (e.g., firewalls, switches, routers) and application layer (e.g., Web servers, DNS, application servers etc.). It detects anomalies of various kinds, including unusual single events, anomalous event parameters, deviating event frequencies, and suspicious violations of trained event correlations. It can notify operators about discovered anomalies via different channels (e.g., email). ÆCID is designed to support deployment in highly distributed environments, as presented in Figure 2.7. Aminer instances can be deployed on relevant nodes (hosts) in the network. Aminer instances are configured and managed by ÆCID Central.

2.5.3 Method

There are two ways for the AMiner to reveal anomalies within the log messages: (1) by observing deviations of the log lines from the parser model, (2) by identifying log lines which do not follow certain predefined rules. Aminer interprets logs from hosts following a specified model (parser model), a graph which represents events logged by that machine (host). A distinct parser model is defined for each service running on the monitored system. A log message that does not match any available path in the parser model graph is considered anomalous, because it represents an unexpected system event. In other words, AMiner whitelists all paths described by the parser model graph and raises an alert every time a log line cannot be fully parsed. In addition, AMiner uses a predefined ruleset, tailored for particular host to distinguish legitimate events from anomalous ones. Rules include permitted (1) combination of variable values, e.g., username associated with IP/MAC address (2) statistical characteristics of events, e.g., frequency of failed logins within 1 minute. Finally, Aminer produces the report of parsed and unparsed log lines, alerts and triggered alarms. Reports are communicated to AECID central or via email directly to the operator.

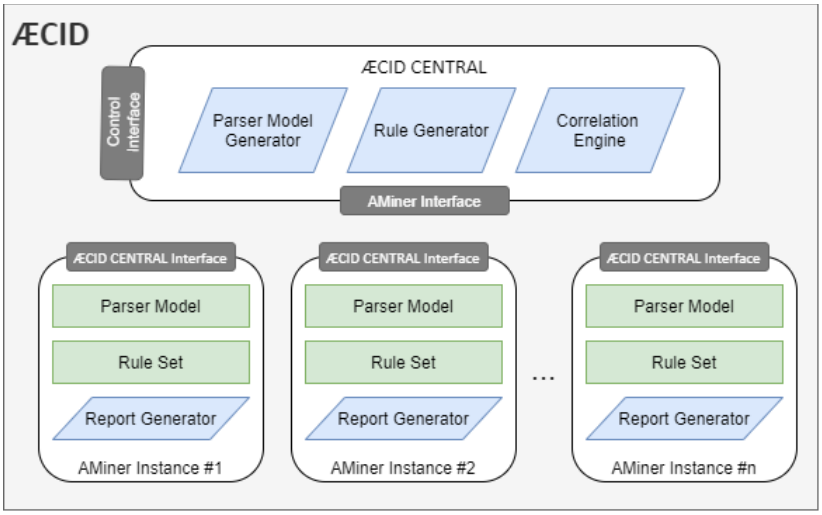

Figure 2.7: Architecture of distributed AECID system (Wurzenberger M., 2018)

ÆCID central provides more advanced functionality, specifically, components that learn the normal system behaviour and define configuration of the Aminer instances (parser models and rulesets). ÆCID central consists of the control interface, parser model generator, rule generator and correlation

engine. Control interface provides information about the deployed Aminer instances, their configuration, status and output (anomalies detected). Based on the log lines which represent normal behaviour of the monitored system, the parser model generator applies word- or character-based clustering to determine static and variable parts of the log lines – from which a graph parser model is derived. Additionally, based on the values of parsed variables, candidate rules are generated. These could be lists or intervals of variable values that are permitted to occur. In addition, ÆCID also supports correlation of events to analyse and associate output of Aminer instances. Aminer can be used as a standalone module to detect anomalies in log data streams. It is required, however, to be provided with manually configured parser models and whitelisting rule sets. The tool is optimized for Linux systems. It can be run from console, as daemon with e-mail alerting and interfacing message queues or embedded as a library into other software solutions. The tool is licensed under the GNU GPLv3 and available via Launchpad[1] or Debian apt-get[2].

2.5.4 Output

The result of the log-based anomaly detection is the list of different anomalies. Depending on the Aminer configuration (types of detector modules used) we can have anomalies associated with the unknown path detected in the system, unknown service, unusual value of the variable (outside of expected range), log frequency anomalies.

2.6 Deep Learning Algorithms for Network Intrusion Detection

2.6.1 Brief Overview

Contributor	AIT
Model/Tool	DL models for Network Intrusion Detection
Data input	network traffic summaries, netflow features
Data description	Features extracted from uni- or bi-directional network flows
Output	Labelled flows (attack, anomaly)
Underlying method	Deep Neural Network (DNN), Deep Belief Network (DBN) and Autoencoder (AE)
Type of learning	Supervised (DNN, DBN) and Semi-supervised (AE)
Explanation	Multi-label and binary classification for attack detection
Labels	Attack labels required for the supervised learning (attack type, anomaly)
Event indicator	Classification score (DBN, DNN) or outlier (anomaly) threshold (AE)
Detection principle	Classification using deep neural networks. Anomaly detection using autoencoders and outlier detection.
Targeted attacks	Network traffic anomalies
Reference	(Rezvy, 2018), (Gao, 2014), (Hawkins, 2002)

2.6.2 Purpose

In order to enrich the detection capabilities offered in SOCCRATES AAD component, we have implemented several state-of-the-art deep learning models proposed for Network Intrusion Detection,

[1] https://launchpad.net/logdata-anomaly-miner

[2] https://packages.debian.org/sid/logdata-anomaly-miner

namely: a Deep Neural Network (DNN) for binary and multi-class classification (Rezvy, 2018), in a supervised and semi-supervised fashion; a Deep Belief Network (DBN) for multi-class classification (semi-supervised) (Gao, 2014); and an Auto Encoder (AE) that is trained to perform outlier detection for unsupervised anomaly detection (Hawkins, 2002).
The algorithms are intended to perform classification based on the features extracted from the network traffic data (NetFlow, packets, traffic summary features).

2.6.3 Methods

Deep Neural Network
To extend the functionality of a regular DNN, which usually works as a supervised classifier (Rezvy, 2018) proposed the idea to stack an Auto Encoder on top of a two-layer classifier network. This algorithm trains the model in three stages: (i) the AE is trained unsupervised; (ii) the classifier is trained with the output of the AE as input to perform a classification, in a supervised fashion; and (iii) the network as a whole is tuned with a few training samples, also supervised. We use this DNN to perform binary classification, to get a comparable result to the AE-based outlier detection (see below).

Deep Belief Network
A model with a similar approach to the AE-DNN is a Deep Belief Network, which is a perceptron network that consists of several layers of Restricted Boltzmann Machines (RBMs). RBMs are often used for dimensionality reduction or feature extraction, which is similar to the algorithm proposed by (Gao, 2014). The model here consists of several layers of RBMs with a single-layer classifier stacked on top. The RBMs are trained unsupervised, the classifier then supervised, using the output of the final RBM as input. The idea is that the RBMs in sequence reduce the input dimension to achieve a better representation of the features.

Autoencoder
(Hawkins, 2002) proposed the use of an AE to perform outlier detection. The approach is to train an AE on benign data (train the network to reproduce the input, unsupervised) and then use the reconstruction error to measure for outliers. The Outlier Factor (OF) of a record i is defined as:

$$OF_i = \frac{1}{n}\sum_{j=1}^{n}\left(x_{ij} - o_{ij}\right)^2$$

Where n is the number of features, x_{ij} the jth feature of record x_i and o_i is the jth feature of the ith reconstructed record. To get a more general outlier detection, we extended this concept with a method proposed by (Azami, 2016), which converts distance-based outlier detection methods to probabilities using a sigmoid function:

$$P(i\ is\ outlier) = (1 + \exp\left(\frac{(OF_i - \gamma)}{\sigma}\right)^{-1}$$

where γ is the anomaly threshold, ($OF_i \geq \gamma \Rightarrow P(i\ is\ outlier) \geq 50\%$) and σ is a scaling constant. We define the threshold as the 95-percentile and as the standard deviation of the outlier factors in the training dataset.

2.6.4 Output

Supervised methods will provide information of detected attack label (multi-label classification). As an alternative we could think of enriching the output with list of associated IP addresses (hosts) or listing the classification confidence per each label. Anomaly detection method (AE) will generate event based on the distance-based detection (binary output). Context information here could be the source IP (or list of associated IPs).

2.7 Autoencoder

2.7.1 Brief overview

Contributor	TNO
Model/Tool	Autoencoder
Data input	NetFlow data
Data description	This tool can use raw NetFlow data records, as well as aggregated NetFlow data or additional DNS related features.
Output	Local hosts with high reconstruction error.
Underlying method	Deep autoencoder
Type of learning	Unsupervised
Explanation	A deep autoencoder is trained to reconstruct expected behaviour of hosts.
Labels	Can be used to create a baseline for normal behaviour but are not necessarily required to detect anomalies.
Event indicator	User-defined threshold for the reconstruction error of the autoencoder
Detection principle	A host is marked anomalous if its computed anomaly score exceeds the user-specified threshold.
Targeted attacks	Network flow anomalies
Reference	(Verhees, 2019)

2.7.2 Scope

The autoencoder was designed as a deep learning-based technique for threat hunting. The motivation behind it was incapacity of out-dated signature-based methods to detect modern and newly emerging cyber-attacks. Therefore, the main role of this tool is to detect statistically anomalous local workstations in the internal network, thus acting as an additional filtering layer of the NetFlow data. This is complementary to the existing defence methods and reduces the amount of data samples that need to be further investigated by SOC analysts.

2.7.3 Data pre-processing

The dataset consists of bidirectional flows, meaning that if an internal host is involved in a flow, the host can either be the source or the destination of the flow. Therefore, a selection of flows is made based on two criteria. Firstly, either the source IP address or the destination IP address of a flow should be equal to one of the internal hosts. Secondly, the internal host should be active in the flow, meaning that the number of bytes sent by the internal host cannot be zero. After the selection is made,

standardisation is applied to scale all features, using a robust scaler (removing the median and scaling the data according to the quantile range), and one-hot encoding is applied to categorical features. Finally, all flows are aggregated per internal hosts for a certain time period (this can range between 15 min to entire days depending on the application and dataset size).

2.7.4 Method

An autoencoder (AE) is a type of feed-forward neural network that is trained with unlabelled training data to produce reconstructions that are close to its original input. By looking at how good an AE can reconstruct a given input, samples that deviate from a certain base line can be identified. To be more precise, measuring the difference between the original input and the reconstruction leads to a reconstruction error, which can be used as an anomaly score. A schematic representation of the autoencoder is shown in Figure 2.8. This example shows an autoencoder consisting of five hidden layers (two of which are needed for encoding the input data, a bottleneck layer that creates a compressed representation of the input, and two layers needed for decoding and restoring the initial dimension to the output).

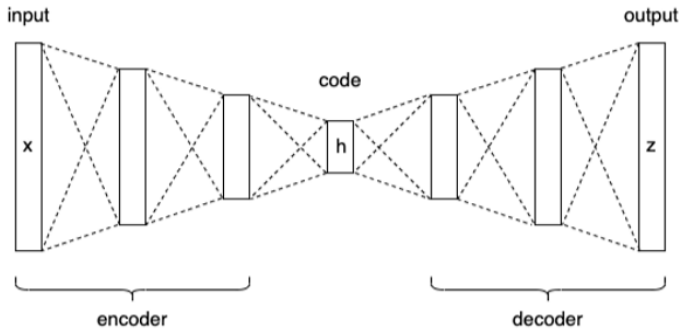

Figure 2.8: Autoencoder architecture

Determining the right number of hidden layers, the number of neurons in these layers and assigning suitable values to hyperparameters such as activations functions, optimizer etc. is done by means of 5-fold cross validation. Finally, the reconstruction error for each data sample is calculated as the mean absolute error between the original data and the reconstructed input.

2.7.5 Output

Currently, the AE has no user-interface and the only output that it displays is a file in csv format with the complete list of local hosts sorted in decreasing order of their reconstruction error, interpreted directly as an anomaly score.

3 Event-based Reasoning

In order to support situation awareness and decision making (incident response) at the security operations center (SOC), solutions are proposed to automatically process output from intrusion detection systems. These solutions are developed to address the following challenges:

(1) Monitoring results in **vast amount of security data (events**) collected from the deployed security sensors, such as intrusion detection systems (IDS), antivirus, firewalls, etc. Given the amount of data and their heterogeneous nature, a manual analysis is not feasible.
(2) **High number of false alarms** reported by intrusion detection systems, which are configured to prioritise reduction of false negatives (minimising the number of missed attacks) at a cost of increasing the number of false positives.
(3) Dealing with **heterogeneous and unstructured data sources**. Different security sensors (detection tools) output different qualitative events that say something about either a local host, a domain or a process. The main challenge of that task is finding a method to associate information sources over different aspects, e.g., reports over hosts and reports over domains.

To address these challenges, substantial body of work has been proposed in the field of alert (event) correlation[1] and sensor information fusion (Salah, 2013). The objectives of the most event (alert)[2] correlation techniques are to: (1) reduce number of false positives, (2) improve understanding of the situation through more holistic approach, i.e. perform reasoning using different type of events to understand what they collectively mean, (3) prioritize events – such that the most severe indicators can be addressed first, and (4) connect current events with previous – sequential patterns of events might be helpful to determine potential next steps of attacks.

According to (Salah, 2013), the main tasks of automated alert processing are alert reduction, prioritisation, aggregation, classification and severity analysis. These tasks can be achieved by means of different correlation methods, which could be roughly divided into similarity-, sequential- and case-based methods. The **similarity-based correlation** is mostly used to reduce number of alerts which stem from the same root cause (attack) and could be considered redundant. The alerts are clustered based on their similarity in terms of attributes (features) as well as temporal aspects. The **sequential-based methods** aim to leverage causal relationships between alerts and usually require definition of pre-conditions and consequences. Using causal relationships are very useful for modelling attack scenarios, natural way of expressing attack steps. Under the category of sequential-based correlation we can find pre-post conditions, graph models, e.g. Bayesian networks, Hidden Markov Models, or more recently Recurrent Neural Networks (RNNs). Finally, the **case-based methods** rely on knowledge based in form of well-defined scenarios (cases) that describe attack. The scenario templates can either be derived from experts or inferred by using machine learning or data mining. The former can pose a potential difficulty in dynamic systems (require manual change of attack signatures). The latter requires representative amount of labelled alert data to derive useable information.

In the SOCCRATES project we aim to leverage output from the anomaly/attack detection tools (described in Section 2) and focus on the sequential-based event correlation techniques. Although alert correlation is usually associated with SIEM functionality, the solution developed here is not intended to replace SIEM (or be considered yet another SIEM solution). Our approach is to keep the event-based reasoning as a standalone service that could be included at SIEM level, but also could be deployed as

[1] Correlation, here, denotes techniques to derive any type of knowledge from the collection of events.
[2] In this document, alert and event are used interchangeably and denote the output of the detection tools.

a sensor in distributed systems. Furthermore, most of the SIEM solutions still to a high extent rely on the rule-based attack detection (case-based alert correlation). Our solution is meant to enable flexible integration of the existing rule-based systems with novel research-based methods.

In what follows, we describe a REASENS framework that is intended to facilitate the integration of the anomaly detection tools with event-based reasoning. Subsequently, we present an approach to integrate events from multiple data sources, i.e. Evidential Networks, which proved to be applicable for handling unreliable data sources (and contradictory evidence). Finally, we will summarise integration of tools in SOCCRATES project.

3.1 REASENS Framework

REASENS Framework is a hierarchical REAsoning system that enables the collection of events from distributed and heterogeneous SENSors. A brief overview of the framework is presented in Table 3.1. The REASENS framework is based on micro-service and event-driven architecture which integrates different security sensors and enables more holistic reasoning about the monitored system state (situational awareness). Microservice architecture facilitates integration of independent (standalone) applications, written in different programming languages etc. The event-driven architecture enables dynamic update of the system whenever something changes without necessity to periodically query the state of all subcomponents.

Table 3.1: REASENS Framework Summary

Contributor	AIT
Model / Tool	REASENS Framework (hierarchical REAsoning system that enables the collection of events from distributed and heterogeneous SENSors)
Data input	Security events (alerts)
Data description	Security events with timestamp, source, type, host etc.
Output	Probabilities of system being in a particular state (under attack, normal operation, erroneous operation)
Underlying method	Aim of the framework is to perform reasoning on collected security events, e.g. root cause analysis, alert correlation and system state inference. In the current implementation the Evidential Networks (ENs) is used (as an alternative to Bayesian reasoning). An EN is a graph structure (hypergraph) that encodes knowledge about variables in a system and the relationships between them. The focus will be on addressing the uncertainty of the data provided by the sensors (e.g. contradictory evidence due to the detection rates of the anomaly detection).
Type of learning	--
Event Indicator	Probability of *Under Attack* state
Detection Principle	Probability of *Under Attack* state is higher than predefined threshold
Reference	(Friedberg, 2017)

Figure 3.1 depicts an outline of the framework architecture. Applications, such as security sensors, are distributed across the strategic critical points in the infrastructure to monitor system's health (behaviour). The framework has been developed to support cyber-physical system awareness hence to

incorporate sensors which monitor both the ICT infrastructure (hosts, applications, network) as well as the underlying physical processes.

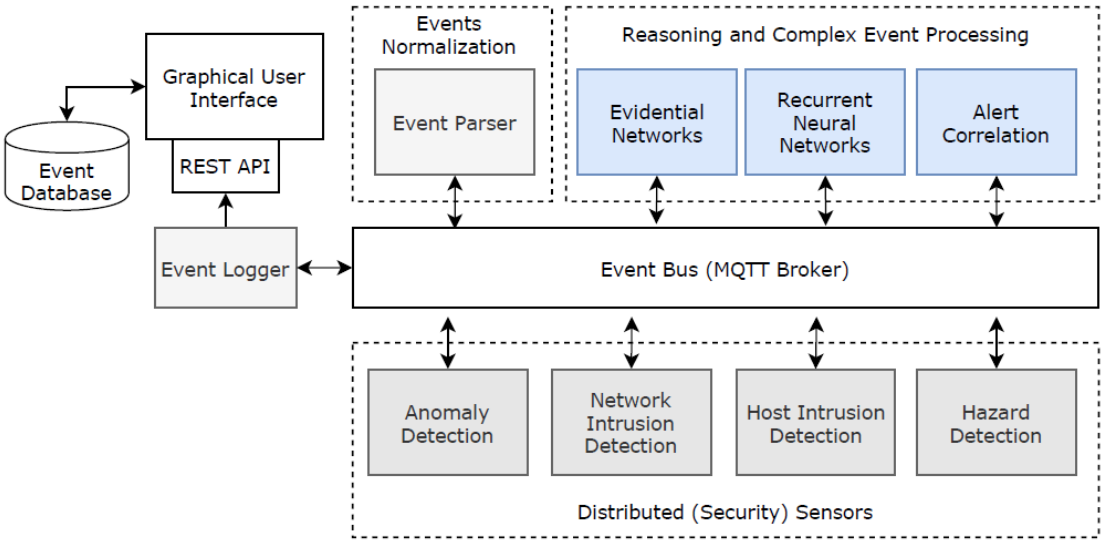

Figure 3.1: REASENS Framework Architecture

Examples of security sensors include anomaly detection in host and network activity, network intrusion detection to check if the network has been manipulated; host intrusion detection system to alert on suspicious activity within the hosts. In addition, sensors to ensure system safety could be deployed locally to control physical process sensors measurements (anomaly detection) or to check whether control commands are safe given the current system state (hazardous control detection).
In current implementation, sensors communicate over MQTT protocol with a more centrally located component (server), which performs high level alert correlation based on the input from the security sensors. Sensors publish events to the MQTT broker, which are then normalized by parser and published further to reasoning and complex processing components, e.g. Evidential Network, Recurrent Neural Networks or other alert correlation algorithms.
The normalised events as well as raw messages from the sensors can be logged into the database and accessed via graphical user interface (GUI). GUI offers web interface (dashboard) with presentation of the system state, logging of the security events, alarms (generated by engines), register and configure sensors etc. The REASENS architecture allows for different types of sensors and reasoning engines which could subscribe to messages sent from selected security sensors. In current implementation, the Evidential Network is deployed as a reasoning engine.

3.2 Reasoning with Evidential Networks

One of the challenges of the event-based processing task is handing incomplete or incorrect evidence from the information sources. Intrusion detection tools operate with different reliability, and it is possible that they will miss some attacks or report false alarms. To address the uncertainty introduced by security sensors, we apply the theory of evidence, also called Dempster-Shafer (DS) Theory, a mathematical framework for data fusion, introduced by (Dempster, 1967) and extended by (Shafer, 1976) and more recently by (Smets, 1990) (Transferable Belief Model). The DS Theory has been recognized as an effective tool to represent different types of imperfections, such as incomplete, imprecise,

uncertain information. The idea is to combine multiple imperfect data sources (information) to reduce uncertainty by collective, coherent integrated information.

3.2.1 Dempster Shafer (DS) Theory of Evidence

In DS Theory, a problem domain, formally called *frame of discernment*, is described by a finite non-empty set $\Theta = \{H_1, H_2, \dots, H_n\}$ of mutually exclusive hypotheses representing all possible solutions of the considered problem (task). Unlike in probabilistic (Bayes) theory where probabilities are assigned to elements of Θ , the DS counterpart, a *basic belief assignment* (*bba*), also referred to as *mass function*, is a function of mapping from power set of Θ, $m: 2^{\Theta} \rightarrow [0,1]$, which satisfies the following conditions: $\sum_{A \in 2^{\Theta}}^{m(A)} = 1$ (mass functions assigned to all subsets of Θ sum to unity) and $m(\emptyset) = 0$ (belief of empty set is zero). The support degree of the proposition *A* is denoted by $m(A)$.

We distinguish two types of ignorance that can be expressed in DS: *global ignorance*, where mass function is assigned to exactly whole set Θ and *local ignorance*, which include all non-singleton subsets of Θ. If there is no local or global ignorance, a mass function reduces to classic probability function.

In addition, the uncertainty captured by DS theory can visualised by Figure 3.2. Belief $Bel(A)$ represents all massess assigned exactly to A and its smaller subsets, and Plausibility $Pl(A)$ denotes all masses that could be assinged to A and its smaller subsets. $Bel(A)$ and $Pl(A)$ can be interpreted as the lower and upper bounds of the real probability $P(A)$.

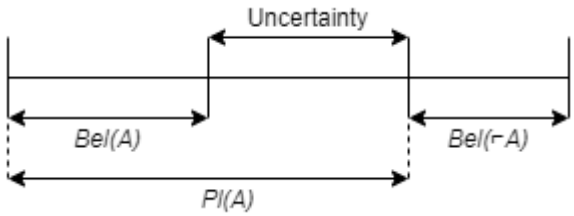

Figure 3.2: Uncertainty with respect to Belief and Plausibility

Dempster's Rule of combination

Fusion of information from sensors is facilitated by the means of Dempster's Rule of Combination. Given two mass function obtained from two independent sources, the combined mass function $m_{\{1 \oplus 2\}}(A) = m_1(A) \oplus m_2(A)$

$$m_{\{1 \oplus 2\}}(A) = \begin{cases} \dfrac{\sum_{\substack{B,C \in 2^{\Theta} \\ B \cap C = A}} m_1(B) * m_2(C)}{1 - \sum_{\substack{B,C \in 2^{\Theta} \\ B \cap C = \emptyset}} m_1(B) * m_2(C)} & , \quad A \in 2^{\Theta} \text{ and } A \neq \emptyset \\ 0 , & A = \emptyset \end{cases}$$

The denominator represents the **conflict coefficient**, a degree of conflict between two mass functions $m_1$and m_2. The combination rule is commutative and associative, which allows to aggregate evidence in any order without changing the outcome. Moreover, the rule is non-idempotent, meaning that

combination of two similar independent mass functions gives generally more precise combined mass function. Combination may increase total amount of information, hence reduce uncertainty.

3.2.2 Evidential Networks

Evidential network (EN) is a graphical model for knowledge representation and reasoning based on Dempster Shafer (DS) theory of evidence (Shafer, 1976), which allows to compute the degree of belief of an event, given evidence. Evidential networks are very suitable for reasoning about complementary evidence from different sensors, and has also shown applicable to reason about the causality between low level evidence and high level system states (Friedberg, 2017).
More formally, EN is a graph structure (hypergraph) that can be described as follows:

$$EN = \{V, \Theta, M, \oplus, \downarrow\}$$

EN encodes knowledge about system variables V (e.g. host state), their frames Θ, i.e. values the variables can take (e.g. normal or compromised) and the relationships between these variables, in the form of mass (belief) functions M. The causality between variables are expressed as mass functions and quantified with a certain degree of confidence. These are propagated through the network to estimate the belief about the root nodes (e.g. system state). Inference is achieved by the evidential operators, called combination ($\oplus$) and marginalisation ($\downarrow$). For decision making, the mass functions of root nodes are transformed into a *pignistic* probability, which is a counterpart of Bayesian probability in theory of evidence.

Figure 3.3 depicts an example evidential network structure, which has been applied to reason about the network node status, based on alerts from different security sensors, such as Antivirus, HIDS, ARP Src, NIDS. The variables representing sensors are usually leaf nodes in the evidential network. The output of these sensors is mapped to other variables, e.g. HIDS maps to detected privilege escalation. The EN nodes between sensors and root node are derived from domain-experts knowledge. The edges in the graph represent mass functions (depicted in green).

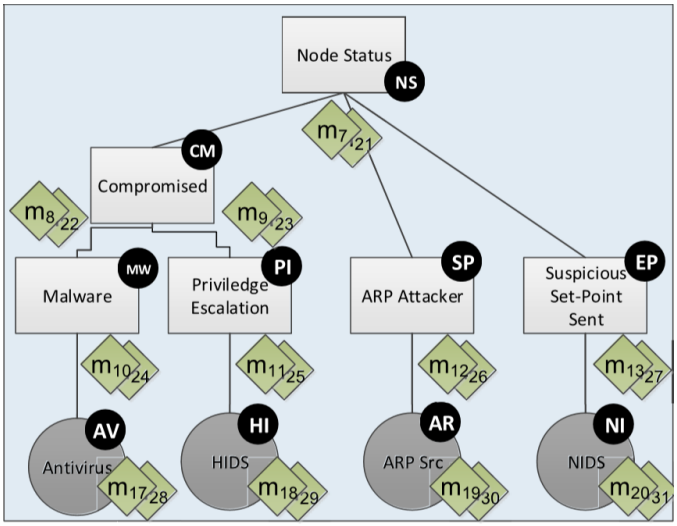

Figure 3.3: Evidential Network Example (Friedberg, 2017)

3.2.3 Sensors' reliability assignment

Combining information from different imperfect data sources may result in contradictory evidence due to different detection rates of the security sensors. Therefore, it is important to represent the knowledge about the sensors' reliability while combining their output in the evidential network. EN allows to represent the uncertainty introduced by the sensors. Different approaches have been proposed in related work (Zomlot, 2011) (Ou, 2009) (Cherifi, 2017), including use of discounting factor based on the reliability score of the sensor. Such approach is only applicable dealing with symmetric reliability, i.e. false positives are as probable as false negatives. Here, we use solution proposed in (Friedberg, 2017), where reliability of a specific state of sensor variable x is modelled by a set of relation implication rules from domain D to D'. These domains represent the same system states but under consideration of reliability. *A-priori* knowledge about sensor reliability can be transferred to EN framework from sensor reliability metrics (e.g. ROC). Since that knowledge is given in classical probabilities, a transfer function is required to express this knowledge as mass functions.

$$P(x_i|\tilde{x}_i) = \alpha$$

$$\tilde{x}_i \subseteq \Theta_D \implies \tilde{x}_i' \subseteq \Theta_{D'} \quad with\ \rho \in [\alpha', 1]$$

To evaluate reliability of the sensors, true positive rate (TPR) and false positive rate (FPR) metrics are used, since they represent conditional probabilities under empirical test results.
We consider TPR and FPR for each potential variable state x_i individual and calculate the positive predictive value (PPV) as follows:

$$P(x_i|\tilde{x}_i) = PPV(x_i) = \frac{TPR(x_i)}{TPR(x_i) + FPR(x_i)}$$

The PPV is interpreted as a conditional probability of state x_i, given that the sensor reports state $\tilde{x}_i$.

3.2.4 Derivation of mass functions

The causality between variables (events) is described by relation implication rules. Domain experts would usually express them as "if A then B" expressions. In evidential networks, these implication rules can be characterised with uncertainty and assigned a certain degree of confidence:

$$A \in \Theta_{D_A} \implies B \in \Theta_{D_B} \quad \text{with } \rho \in [\alpha, \beta], 0 \leq \alpha \leq \beta \leq 1$$

These rules are then mapped to mass functions, as described in more detail in (Friedberg, 2017). Nevertheless, collecting knowledge (mass function estimates) from domain experts, is a manual and static process. Given a sufficient amount of training data, the causal relationships can be discovered by data analysis. Our future work intends to investigate potential learning algorithms that would allow evidential network to derive mass functions from training data.

3.3 SOCCRATES tools integration

Figure 3.4 depicts an overview of the SOCCRATES approach to event reasoning component. Multiple detection tools are available from SOCCRATES partners (as described in Section 2). These detection tools generate events based on heterogeneous data sources, including network traffic, raw log data from network nodes and hosts, as well as logs from security tools such as firewalls, web proxies, DNS systems, vulnerability scanners, antivirus and IDS/IPS. Detection tools provide information about the monitored system state, e.g., *abnormal activity at host X*, *network attack detected*. The output from the tools is used for event-based reasoning to answer questions about the system state, e.g., question about host being compromised, or the root cause of the reported anomalies. Another potential task that could be supported by event-based reasoning is to distinguish between anomalies caused by cyber-attack and benign anomalies, e.g. due to changes in configuration. The parts of the diagram depicted in green are methods developed specifically for the project. It is intended to use events from all available detection tools as well as additional context information from other SOCCRATES platform tools, e.g. obtained threat intelligence (input from Threat Collection and Prediction (TCP) Component).

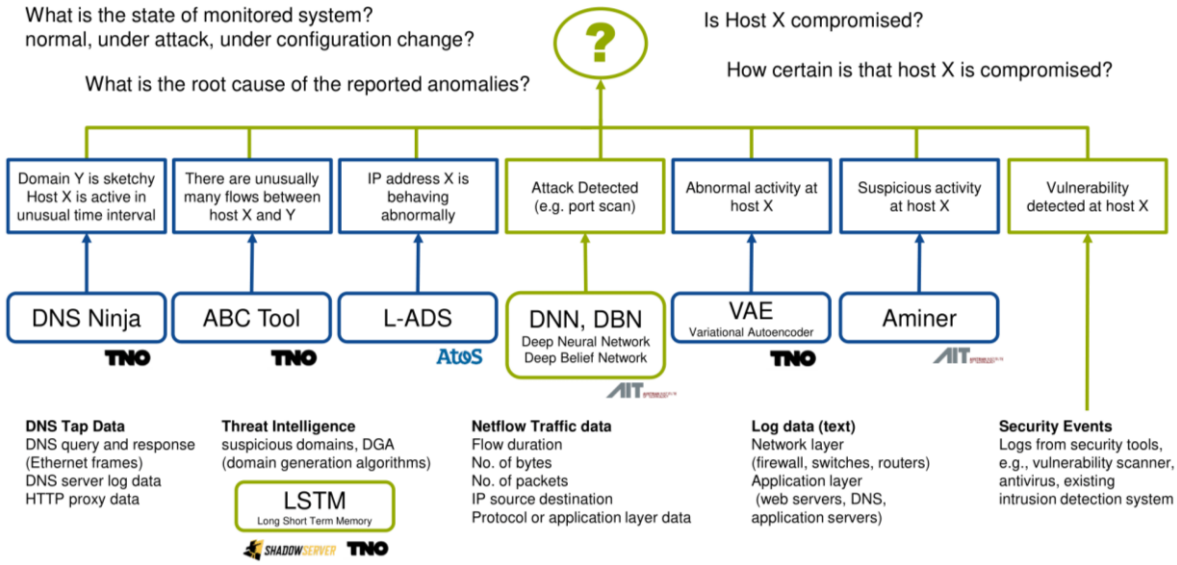

Figure 3.4: Overview of SOCCRATES detection tools and potential reasoning objectives.

The work developed in T4.2 uses (big) threat intelligence data collected by Shadowserver to perform detection of suspicious domains, likely to be automatically generated by Domain Generation Algorithms (DGAs). The output of LSTM (detection algorithm) is envisioned to be included as a data source or wrapped as a new detection sensor. More details concerning the technical implementation of the DGA detection algorithms can be found in Deliverable D4.2.

3.3.1 Data pre-processing

To support efficient processing of the events and as well to ensure the robustness to increasing number of sensors (scalability), normalisation and filtering of events is assumed. The first one relates to translating events to a common format to enable further processing in reasoning engines. A potential event format could be IDMEF (Intrusion Detection Message Exchange Format) adapted to JSON

message format, which could be communicate via MQTT protocol. Event filtering refers to clustering events that refer to the same location in the network (host or cluster of hosts).

3.3.2 Reasoning Objectives and Evidential Network Structure

As mentioned earlier, there are different objectives for holistic event-based reasoning. We intentionally do not specify one concrete objective for the proposed evidential network. The determination of the objective and evidential network structure to high extent rely on the available data and characteristics of the events generated by the detection tools. Our intention is to explore different configurations of sensor integration as well leverage additional context knowledge from the monitored system, e.g. input from vulnerability scanner, potential attack paths etc.

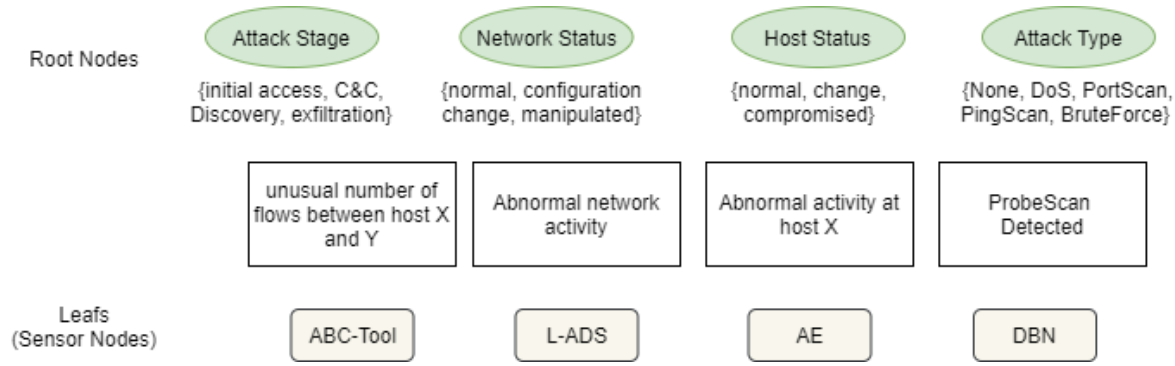

Figure 3.5: Types of nodes in Evidential Network: sensors are represented by leaf nodes (those with no child nodes) and output of the reasoning engine by root nodes (those with no parent nodes).

For now, we list several reasoning objectives that, in our perspective, could be used by other components within the SOCCRATES platform:

1) Determining the attack stage could be helpful to select appropriate response action, e.g.:
   - Initial access can be handled with immediate containment,
   - Lateral movement requires assessment of scope of compromise to perform containment
   - Dealing with exfiltration and web application-based attacks, the first step would be to stop exfiltration, and then continue with the other steps.
2) Network status and Host status information could be useful for attack localisation and scope assessment. In addition, anomalies that are not caused by malicious actors, e.g. changes in host behaviour, change of network configuration, new hosts in the network, could also be determined using additional context information from infrastructure modelling tool.
3) Reasoning about the attack type could support attack detection reported by different intrusion detection system. Structure of the network and information captured from the sensors should reflect IOCs (indicators of compromise) typical for specific attack.

3.3.3 Advanced Attack Detection

Ultimately, the objective of the AAD component is to support detection of advanced attacks. As specified in WP2 (Eian, 2019), the AAD component is involved in Use Case 1 (UC1) and is intended to handle different **attack types**, such as ransomware attack on a single system or multiple systems, APT style espionage, web application attack with focus on personal data theft.

To ensure a more standardized description of these attacks, they are expressed within the MITRE ATT&CK framework[1], as follows:

- Scenario 1.1: Detection of Initial Access / Execution / Command & Control
- Scenario 1.2: Detection of (internal) Discovery / Lateral Movement
- Scenario 1.3: Detection of Exfiltration (of confidential data and/or personal data)
- Scenario 1.4: Detection of web application attack (with personal data theft)

Given the purpose specified for each tool (see Section 2), we have summarised their capabilities with respect to the MITRE ATT&CK tactics, as shown in the table below.

Table 3.2 Detection tools capabilities with respect to MITRE ATT&CK Framework

ATT&CK tactic	ABC Tool	DNS Ninja	L-ADS	NIDS (DNN,DBN)	(V)AE	Aminer
Initial Access	✓					
Execution				✓		✓
Command & Control		✓				
Lateral Movement	✓					
Internal Discovery	✓		✓	✓	✓	
Exfiltration		✓	✓		✓	
Privilege Escalation				✓		✓

For example, ABC tool models the internal network behaviour, therefore any unusual incoming connections could indicate *initial access*. ABC tool also reports anomalies with respect to known hosts communication, e.g. abnormal amount of connections between two hosts or irregular connections between machines that did not communicate earlier. These events could indicate the *lateral movement* or *internal discovery*.

Anomaly detection tools usually report anomalies based on some sort of score, a numerical value that expresses the deviance of the observed sample from the normal system behaviour (e.g. distance measure in clustering approaches or reconstruction error in autoencoder). It is reasonable to assume that such an anomaly score could be used to prioritize events in SIEM, based on the assumption that deviation from normal behaviour is correlated with the severity of the event. Nevertheless, there are situations when this assumption does not hold, and the attack is only detected by so called weak indicators (anomalies with low confidence scores). Therefore, reasoning about the events collectively could discover the meaning and importance of many weak indicators, which could easily be overlooked when

[1] https://attack.mitre.org/

analysed individually. Evidential network (REASENS) aims to combine different indicators to enhance the detection and increase the confidence of reported events.

As depicted in Figure 3.6, ABC tool can produce evidence that support tactics, such as *initial access*, *lateral movement* and *internal discovery*. The anomaly scores of reported events may vary depending on the parameters of the two latter tactics (attacks), such as frequency and magnitude of queries in network scans. Even if ABC tool reports a "weak" anomaly (i.e. anomaly with low score), the presence of other evidence, e.g. anomalies reported by anomaly detection tools (L-ADS, AE), or suspicious activity detected at source host (Aminer) could increase the confidence in detection of an ongoing attack (*internal discovery*)

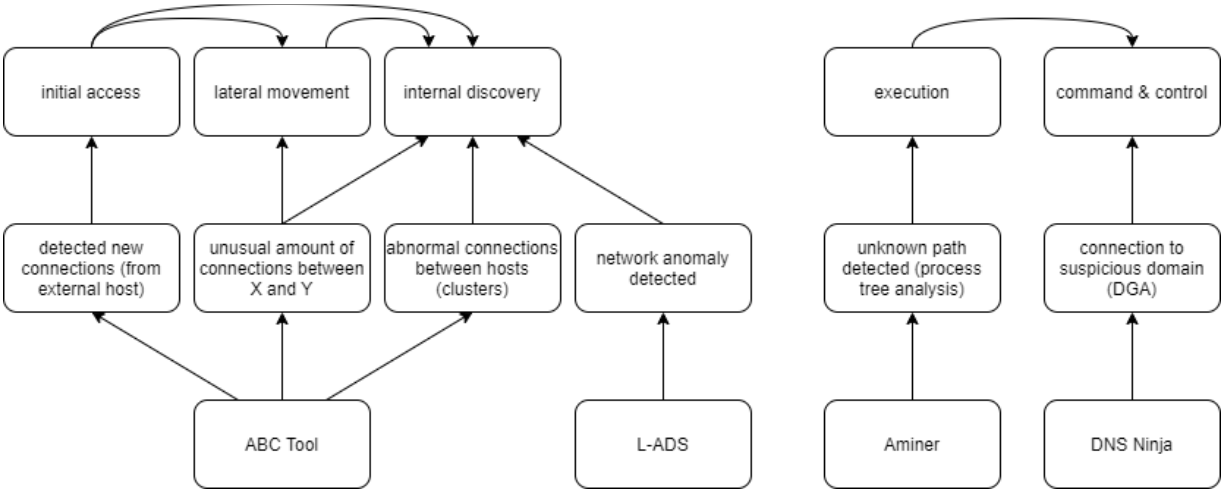

Figure 3.6 Example of evidential network for reasoning about MITRE ATT&CK tactic (stage).

Another example is based on potential approach to handling the *Scenario 1.1: Detection of Initial Access / Execution / Command & Control.* We assume that the attack is first detected by the ABC-tool with a weak indicator of one irregular connection being made to the host (*initial access)*. This event would typically be of rather low score. Then, at the same host Aminer detects unusual services being started (unknown path in process tree analysis) which might suggest that the host is being compromised (*execution*). Further, the DNS Ninja reports an anomaly about connection to suspicious domain (DGA), which indicates *command & control.* Despite of the value of each of the above-mentioned indicators, the combination of all of them can improve the detection of an ongoing attack with some knowledge about the attack step that is currently being performed.
In addition, the causal sequence of the attack steps could be modelled within the evidential network, as shown in Figure 3.6. If the above-mentioned *Scenario 1.1* events are followed by alerts from DNS Ninja together with anomalies from L-ADS, AE and other network anomaly detection tools – the confidence of attack being an *exfiltration* is respectively higher.
Another example could be when Aminer reports a potential *Execution*, and that is followed by DNS Ninja reporting suspicious domain access, the confidence of *Command & Control* is increased.

The presented causal connections between evidence, tactics and attacks are derived from expert's knowledge or collected threat intelligence (MITRE ATT&CK framework). Nevertheless, we expect to discover other dependencies from the actual training data, based on the detection performance of individual tools. Since the DNN and DBN methods rely on supervised training, given an appropriate training set we could fill the gaps and address the limitations of the other anomaly detection tools.

3.3.4 Evidential Network Output

We envision the output message from the reasoning engine in the minimal format, as depicted in Figure 3.7 (could be extended as the tool develops). It is important to keep the message format easy to extend with additional reasoning objective outputs. The output from the EN will be sent further to the Security Monitoring Solutions (SIEM). The information that could be useful for the SIEM and staff at SOC are most likely: the events (from detection sensors) associated with generated EN event and results in terms of the probabilities of each root node in the evidential network.

```
{
   "UniqueID"    :        123456789876543…
   "Timestamp" :          1593741238,
   "Source"       :        "EN",
   "Events"       :        ["123456..", "23435679.."],
   "Results"      :  [
      "Host Status" : {
            "normal"                : 0.139,
            "benign change"      : 0.231,
            "compromised"        : 0.630
      },
      "Network Status" : {
            "normal"          : 0.223,
            "compromised"  : 0.787
      },
      "Attack Type" : {
            "DoS"           : 0.20,
            "PortScan"       : 0.76,
            ...
      },
      "Attack Tactic" : {
            "Initial Access"          : 0.10,
            "Exfiltration"            : 0.76,
            "Lateral Movement"   : 0.03,
            ...
      },     ...
   ]
}
```

Figure 3.7: Example of the output from evidential network

4 Proof of concept

In this section, we present a proof of concept of the proposed methodology for detection tools integration. Our objective is to demonstrate how complementary evidence from anomaly detection can be combined to achieve better understanding of the system state. We investigate how the reliability of the detection tools can be accounted for in the overall detection performance.
In the following, we firstly describe the experimental set-up with respect to dataset and detection tools used. Next, we present the analysis of the results achieved by the detection components and building on that analysis we propose the structure and configuration of the evidential network. Finally, we provide the evaluation of the overall solution and summarise future directions.

4.1 Experimental setup

4.1.1 Tools Overview

The overview of the tools used for the proof-of-concept experiments is shown in Figure 4.1. Initially, our goal was to combine more variety of tools, e.g. network- with host-based anomaly detection. However, our major obstacle was lack of representative, labelled dataset which contains both: network traffic and logs collected during attack simulation. Therefore, we focus on the network-based detection due to availability of benchmark, public datasets for NIDS evaluation. For our experiments we have chosen the CIDDS-01 dataset.

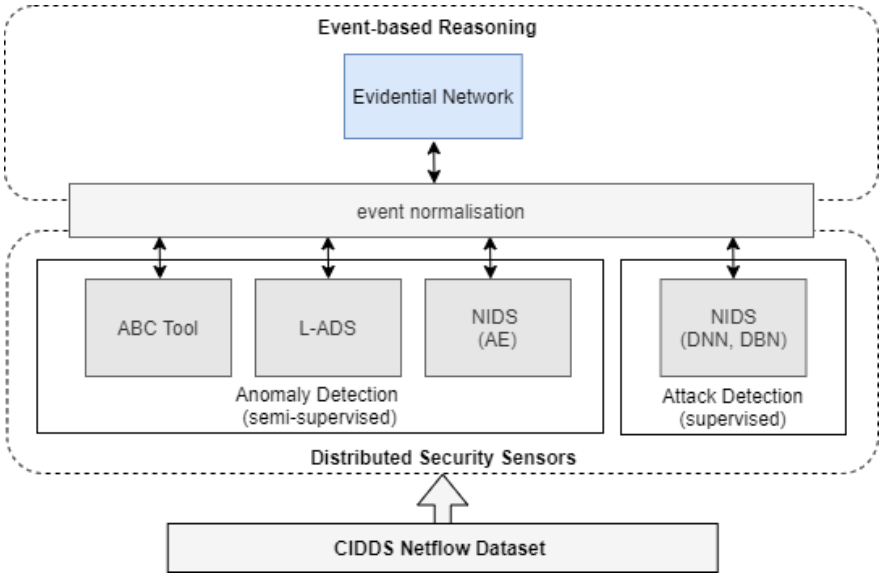

Figure 4.1: Tools used in the Proof of Concept experiments

Since we are bound to one type of data source (netflow), in our approach we try to combine tools, which provide slightly different information about the system. ABC Tool uses netflow to learn the host behaviour and can be used to provide information about hosts being compromised. The anomaly detection (L-ADS, AE) tools label anomalies on the flow basis. To introduce additional variety, we propose to use output from supervised attack detection (DNN and DBN). The events (alerts generated by tools) are provided in the same format – predicted label per each flow in the dataset. The evidential network is used to combine the events.

4.1.2 Data selection

To realize the proof of concept described in this chapter, a suitable dataset was necessary. This dataset had to fulfil the following requirements to allow employing the detection tools shown in Figure 4.1:

- Dataset consists of at least NetFlow records, supplemented ideally by system log data. This is the input requirement for the selected tools.
- Dataset includes information about the network infrastructure and specifically, about which IP addresses correspond to internal hosts. This piece of information is required by the ABC Tool to create the baseline cluster model for the internal network.
- Dataset needs to include labels to allow the computation of performance metrics such as accuracy, precision, recall etc.
- Dataset needs to be sufficiently large to represent network behaviour over a reasonably long timespan. Therefore, data needs to cover at least a week of network traffic to capture the essential temporal patterns.
- Dataset needs to contain various types of attacks. This allows us to assess the detection tools on different attack tactics, visible in different data sources. Ultimately, we are aware that no public dataset can fully encompass the advanced attack threats that are meant to be detected with the AAD. For this reason, in the final deployment, the AAD will be tested by a red team and/or by means of adversary emulation (e.g. Atomic Red Team) at the pilot site against MITRE ATT&CK tactics. We elaborate on this in Section 5.1.

At that point in the project, it was agreed that timewise was essentially infeasible to collect data at one of the pilot sites, which would have in any case lacked formal labels. The alternative was thus to find a publicly available dataset. The starting point was the most recent survey of network intrusion detection datasets by (Ring M. W., 2019). Based on this work, we have selected a few promising datasets, which were matched against our requirements. These are listed in Table 4.1.

Table 4.1: Overview of datasets surveyed in the Proof of Concept

Dataset	Description	Limitation
Kent, 2016 (Kent, 2015)	Labelled unidirectional flows and logs from a real enterprise network spanning 58 days	Download of the log data file failed. So, we could not collect the entire dataset.
NGIDS-DS (Haider, 2017)	Labelled packets and logs from a small, emulated network	The timestamps of the provided packet labels do not match those in the packet capture file. Thus, we cannot ensure correct labelling.
CIDDS (Ring M. W., 2017)	Labelled unidirectional flows from an emulated small business environment spanning 4 weeks	Only a simple range of attacks is considered.

From the data survey two conclusions can be drawn: 1) in practice it is very difficult to find a proper dataset containing both host and network information; 2) the CIDDS (Coburg Intrusion Detection Data Sets) (Ring M. W., 2017) was the only suitable choice since its limitation is not critical in testing the functionality of the proof of concept AAD.

4.2 CIDDS dataset

CIDDS (Coburg Intrusion Detection Data Sets) was proposed for evaluation of anomaly-based network intrusion detection systems. The dataset has been generated using a virtual environment (testbed implemented using OpenStack[1]). The dataset has been created to emulate a small business environment (several clients and typical servers like E-Mail or Web servers). Different behaviour profiles have been implemented to generate normal traffic, such as browsing web, sending and receiving emails, file exchange; the activities are done with respect to time schedule (work time, lunch breaks, etc.). Malicious traffic includes three types of attacks executed within the network: Denial of Service (DoS), Brute Force and Port/Ping Scans. Generated network traffic is captured in unidirectional netflow format and annotated with four types of labels: *(1) attackType* (DoS, PortScan, PingScan, BruteForce), *(2) attackDescription* (more detailed overview of the attack), *(3) attackID* (unique id to identify flows that belong to the same attack), and *(4) class*, which refers to host (normal, attacker, victim, suspicious and unknown). A summary of CIDDS feature is listed in Table 4.2.

Table 4.2: Types of features in CIDDS-01 dataset.

Flow-based	Numerical	Categorical	Labels
Source IP Address	*Duration*	*TCP-Flags*	*class* (w.r.t host)
Source Port	*Bytes*	*Protocol*	*attackType*
Destination IP Address	*Packets*	*Type of Service (ToS)*	*attackID*
Destination Port	*Flows*		*attackDescription*
Date First Seen			

In our experiments, we use a subset of the CIDDS-01 dataset for training and evaluation of the AI-based detection components. To enable comparison of accuracy of different tools, we use the same training and testing data for all algorithms. The choice of the train-test data has been biased towards requirements of the ABC Tool. The goal of the ABC-tool is to create a host behavioural model for a certain interval of time. One can look at the expected host behaviour during a certain working hour interval in a day, during the whole day or even the whole week. Therefore, it is important to have a training set that correctly represents the time aspect. Moreover, the tools only look at communication flows between the internal hosts and leave out all flows to/from outside.

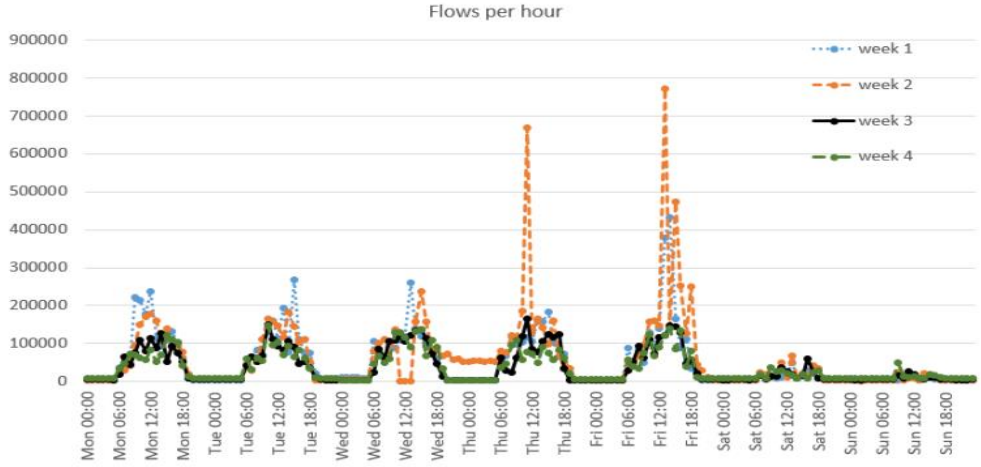

Figure 4.2: Flows distribution over time for all four weeks of the CIDDS-01 dataset (Ring, 2017)

[1] https://www.openstack.org/

CIDDS-01 contains four weeks of network traffic and distribution of flows during each week is depicted in Figure 4.2. To support the behaviour analysis, a week time interval has been chosen for training and testing. Week 4 has been chosen for training as it contains the highest number of normal internal flows. This aspect is important since anomaly detection algorithms are trained only on benign behaviour. The test set has been selected from week 2, as it contains samples from all attacks, as well as network failure (Wednesday at 12:00). Distribution of flows in training and testing set with respect to class are depicted in Figure 4.3.

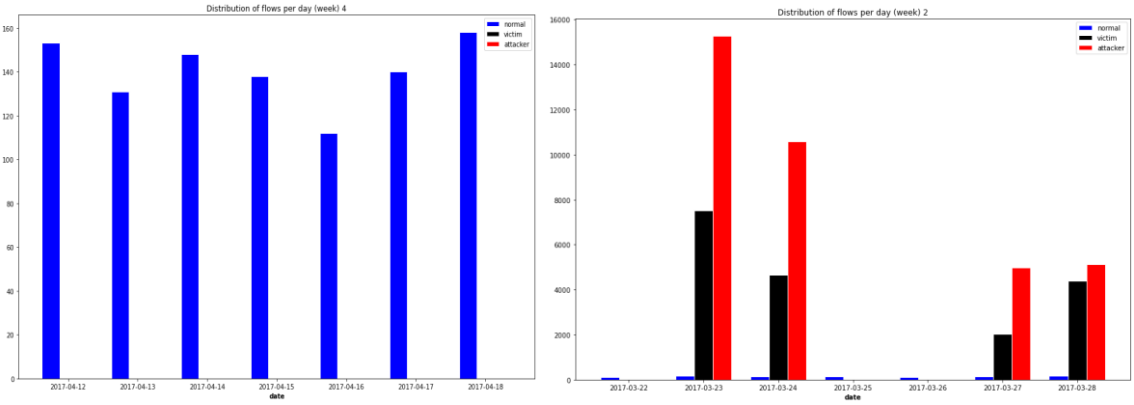

Figure 4.3: Distribution of flows in the training (week 4) and test (week 2) set.

For the evaluation of the evidential network we further split Week 2 into train-test datasets, preserving the distribution of the attacks, as depicted in Figure 4.4. The meaning of training with respect to Evidential network is that we use the results of the tools to calculate their performance metric to parametrise EN. Furthermore, the mass functions (relationships between network nodes) are also partially derived from patterns observed in the training set.

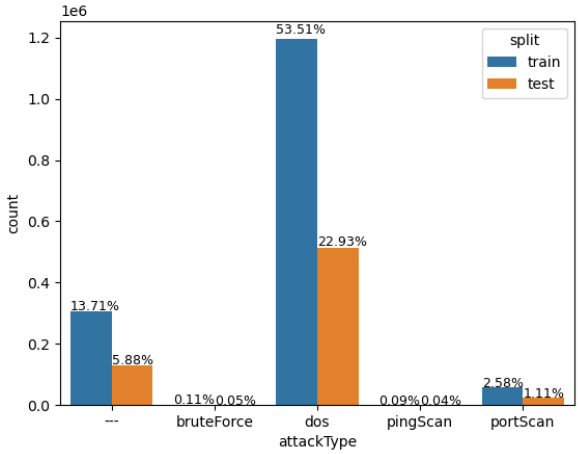

Figure 4.4: Train-test dataset split (week2) for evaluation of evidential network

The test set is used to evaluate the performance of the combined solution. We have decided to make the test set slightly bigger and use the 70-30 split. In the following, we perform flow-based evaluation, i.e. results are reported per each flow. Such approach is enabled by mapping the output of all tools into the same format (flows). Nevertheless, such approach will not be possible with tools which use different data to detect attacks. In further experiments, we will use the attack-instance (attack-ID) based evaluation.

4.3 Anomaly and Attack Detection Results

In the following, we analyse the output of each detection algorithm to gain some insights of strengths and weaknesses of each method with respect to different detection tasks and get intuition on how the results could be improved by tools integration.

4.3.1 Anomaly Detection Results

As mentioned before, the ABC tool uses week 4 to model normal behaviour of the system by clustering internal communication patterns. The model derived in the training phase is then used to detect anomalies by analysing deviations in the model obtained in test phase, as depicted in Figure 4.5.

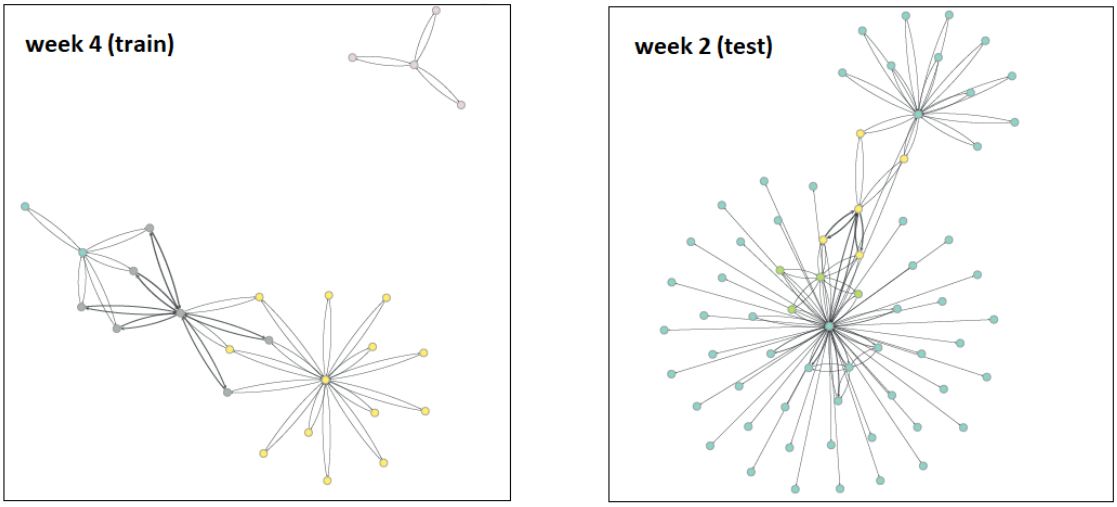

Figure 4.5: Cluster Visualisation for hosts in CIDDS01 dataset

The ABC tool organized hosts (represented by IP addresses) into four clusters:

0 : ['192.168.200.2', '192.168.200.3']
1 : ['192.168.200.8', '192.168.200.9', '192.168.200.4', '192.168.200.5', '192.168.220.16', '192.168.220.13', 'EXT_SERVER']
2 : ['192.168.210.5', '192.168.210.4', '192.168.210.2', '192.168.210.3']
3 : ['192.168.220.14', **'192.168.220.15'**, '192.168.220.12', '192.168.220.10', '192.168.220.11', '192.168.220.4', '192.168.220.5', '192.168.220.6', '192.168.220.7', '192.168.220.2', '192.168.220.3', '192.168.220.8', '192.168.220.9']

Based on modelled activity of those clusters, an unusual behaviour is identified (on the cluster level). The output of ABC tool is a list of hosts, sorted according to the anomalous activity (anomaly score). Each host is described by the additional information, e.g., number of connections to new clusters, connections to new IPs, irregular connections. For example, information for anomalous host is summarised as follows:

Host	192.168.220.15 (cluster #3)
Is Anomalous	1
New cluster-connections	2
Connections to new IPs	41
Irregular connections	0
Anomaly deviations	[(2194.875045475966, 1.0)]
Anomaly cluster list	[0.0, 1.0, 2.0]

New cluster-connection list	[0.0, 2.0]
New IP list	['192.168.200.0', '192.168.200.1', '192.168.200.10', '192.168.200.11', …, '192.168.210.9', '192.168.220.1']
Irregular connections list	[]

ABC tool correctly classified the host (192.168.220.15) as being anomalous. In fact, that's the host from which all the attacks were executed. In the evaluation, the results from the ABC tool were used to label the netflow data as anomalous or not. This is done to enable comparative analysis of tools (flow-based output). The performance of the algorithms is expressed using the following metrics (based on the number of true positives *TP*, true negatives *TN*, false positives *FP* and false negatives *FN*):

1) $Accuracy = TP+TN/(TP+TN+FP+FN)$, which is a proportion of true answers out of all answers
2) $Precision = TP / (TP+FP)$, that is a proportion of true positives out of predicted positives
3) $Recall = TP/(TP+FN)$, denotes a proportion of actual positives that are correctly predicted
4) $F1\ score = 2 \cdot \frac{precision \cdot recall}{precision + recall}$, which is a harmonic mean between precision and recall.

The summary of the performance metrics achieved by the ABC tool, L-ADS and NIDS-AE is listed in Table 4.3. The first column (*) lists results of ABC on the overall week 2 set (including the external server node). The second column is calculated only on the pre-filtered internal flows (aligned with the output of other tools). For the precision, recall and F1 score, the *macro avg* values are used (un-weighted mean which does not account for label imbalances). Additionally, Figure 4.6 shows the corresponding confusion matrix per each anomaly detection tool.

Table 4.3: Evaluation of the Anomaly Detection Tools

metric	ABC Tool *	ABC Tool	L-ADS	NIDS-AE
Accuracy	0.835	0.663	0.952	0.991
Precision	0.750	0.524	0.910	0.987
Recall	0.659	0.822	0.949	0.985
F1 Score	0.701	0.443	0.927	0.986

In terms of accuracy, L-ADS and AE show very good results (higher than 95%) at low number of false positives. Although both solutions (L-ADS and AE) are based on autoencoders, we they apply slightly different pre-processing techniques and input features. It is rather unusual for autoencoders to achieve accuracy of that level, but it might be due to the characteristics of the dataset, e.g. high discrepancy between benign and anomalous samples (high reconstruction error for attack samples).

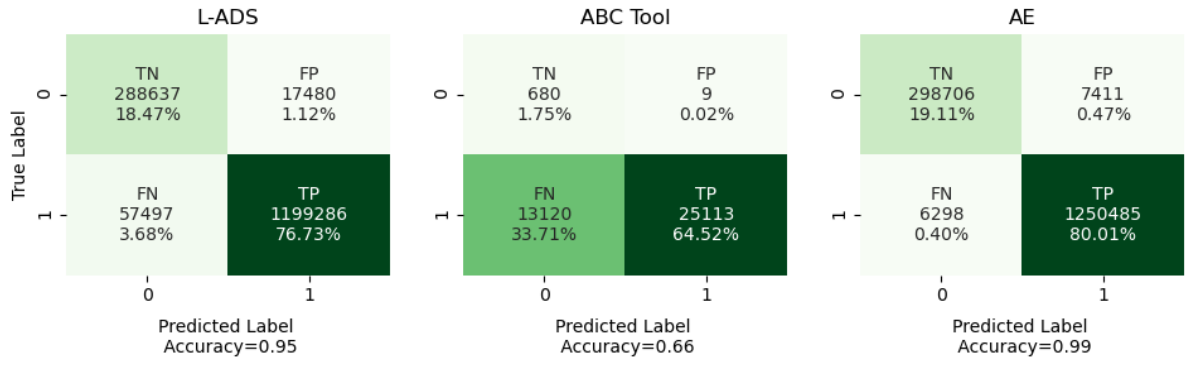

Figure 4.6: Confusion matrix for the anomaly detection tools (L-ADS, ABC, AE)

Furthermore, the low accuracy score of the ABC-tool is misleading as it's mainly caused by the high number of false negatives. The tool identifies flows from the compromised host (attack class), whereas in CIDDS-001 attacks labels denote both: attacker and victim hosts. Since ABC tool does not report the victim nodes – these samples are collected under false negatives (for more apparent visualisation see Figure 4.8).

4.3.2 Attack Detection Results

The Deep Neural Network (DNN) and Deep Belief Network (DBN) are used as supervised models for attack detection. The input features related to IPs are not used for classification (not suitable for characterizing attacks). The categorical features are transformed using one-hot-encoding, i.e. binary feature is used per each category. The resulting input vector contains 29 features. In addition, all features are normalised using min-max transformation. The objective of models is to classify samples according to *attackType* label. Since supervised models require to be trained with attack samples, a subset of week 2 data is used as a training set for DNN and DBN. The results of both classifiers are shown in Figure 4.7 (confusion matrix) and listed in Table 4.4.

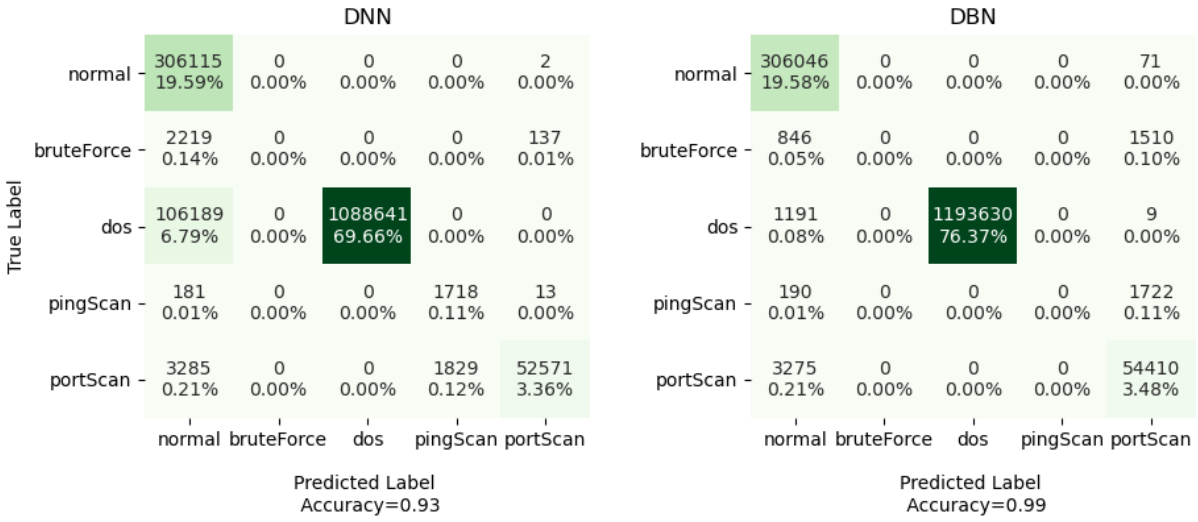

Figure 4.7: Confusion matrix for the attack detection tools (DNN, DBN)

We can observe that both models achieve the best performance (highest number of TP) in DoS attack detection, which is likely due to amount of training samples and distinct characteristics of that attack. From the scan attacks, pingScan and portScan, the latter seem to yield better results. DBN misclassifies all pingScan samples as portScan. Both models failed to detect any of the bruteforce attacks. A potential reason for poor results in bruteforce and pingscan detection is bias in training set (the number of samples of these attacks is lower than 10%).

Table 4.4: Evaluation of the Attack Detection Tools

	DNN	DBN
Accuracy	0.927	0.994
Precision	0.643	0.585
Recall	0.744	0.588
F1 Score	0.676	0.587

The analysis of tools output has two main objectives. First one is to use the performance metrics to access the reliability of the sensors. Second objective is to gain insights on what the strengths and weaknesses of each approach are. For example, as we have learned from the results, the supervised models are biased towards DoS attack detection, whereas pingScan or bruteforce attacks are often misclassified. We analyse the results of the anomaly detection with respect to the *attackType* label. As we can see in Figure 4.8, most of the true positives (TP) in L-ADS and AE results correspond to DoS attack, with slight percentage of portScan (visible in AE plot). Since ABC tool does not address DoS, the resulting true positives are divided between PortScan and bruteForce. One approach would be to leverage the ABC tool results to improve scan or bruteforce attack detection.

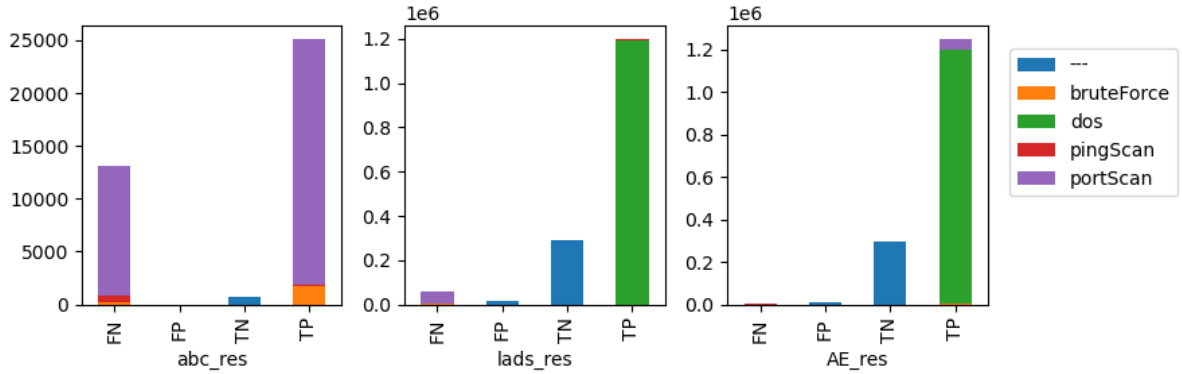

Figure 4.8: Anomaly detection results with respect to attackType label.

Additionally, we analyse the results distribution of all tools with respect to class (host). Potential reasoning objective would be to support evidence provided by ABC tool (about hosts being compromised or not) with results of other tools. As can be observed in Figure 4.9, the ABC tool associates only attacker class hosts as anomalies, hence the high number of false negatives (representing class victim). The other tools show even proportion between attacker and victim class in their true positives.

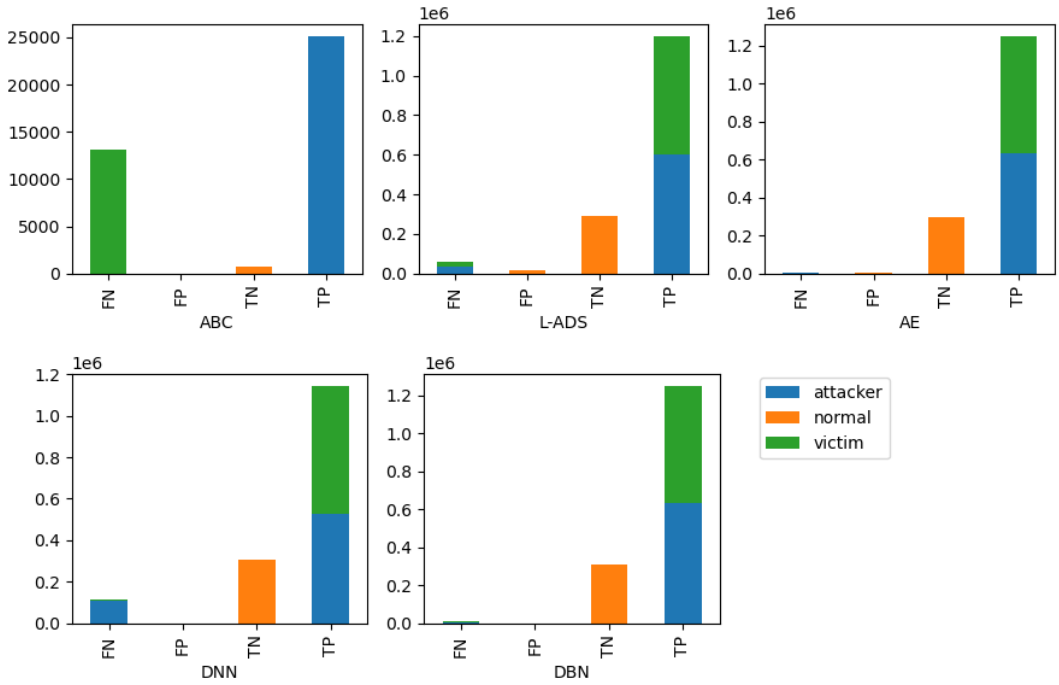

Figure 4.9: Detection tools results with respect to class label.

4.4 Evidential Network Results

The structure of the Evidential Network is shown in Figure 4.10. We propose to combine the detection tools to reason about the state of each internal host (*Host X State – normal, victim, compromised*) and overall system state (*System State – normal, under attack*).

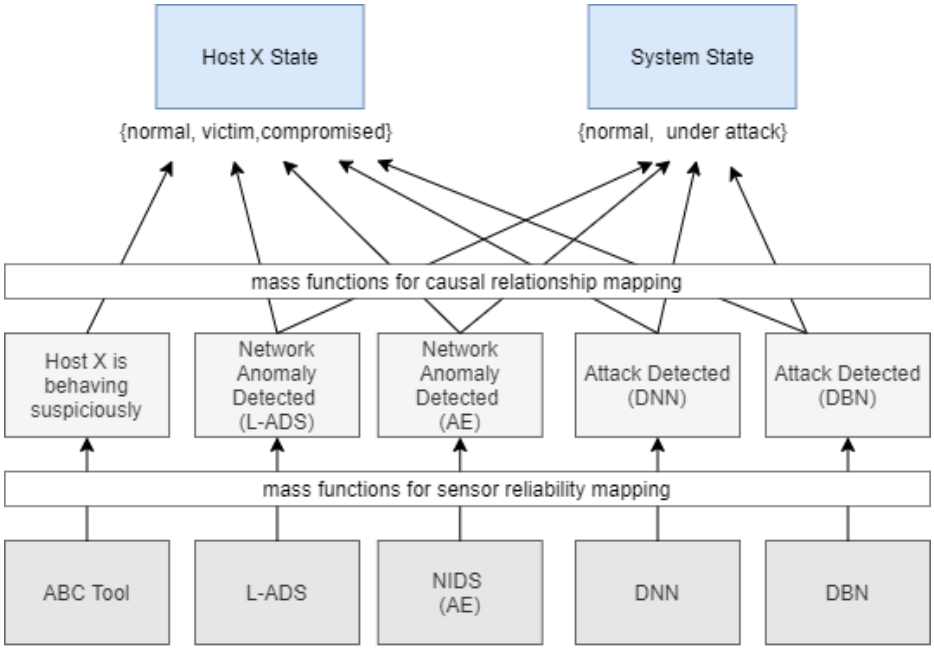

Figure 4.10: Structure of the Evidential Network

As mentioned in Section 3.2.2, evidential network is a graph model where nodes represent variables and mass (belief) functions relationships between those variables. The bottom nodes (sensors) are mapped to the events generated by those sensors with certainty derived from ROC metrics (true positive and false positive rates). Causal relationships between events reported by sensors and the root nodes (host and system state) are based on the observations from data analysis.

Table 4.5: Evidential network configuration: variables and their frames.

Variable	Values (frame)	Descriptive Values
DNN	{0,1}	normal, attack
DBN	{0,1}	normal, attack
L-ADS	{0,1}	normal, anomaly
ABC	{0,1}	normal, anomaly
AE	{0,1}	normal, anomaly
Host {x} behaving suspiciously	{0,1}	normal, anomaly
Anomaly detected (L-ADS)	{0,1}	normal, anomaly
Anomaly detected (NIDS-AE)	{0,1}	normal, anomaly
Attack detected (DNN)	{0,1}	normal, anomaly
Attack detected (DBN)	{0,1}	normal, anomaly
System State	{0,1}	normal, under attack
Host State	{0,1}	normal, victim, compromised

The results of the event based (EN) reasoning are expressed as a (*pignistic*) probability of variable taking particular value, e.g. a probability of *host {x} being in compromised state*. In order to compare the results of EN with other tools, the performance metrics were calculated for EN results. The output for each EN variable is the state with the highest probability, given that it is greater at least by 0.1 than probabilities of other states for that variable. The results of the detection tools and integrated solution, on the test subset of week 2 are listed in Table 4.6.

We can observe that the integrated solution, in most cases yields as accurate solutions as anomaly detection tools (ABC, L-ADS and AE). It improves the precision of recognising the normal system state (reduces number of false negatives in anomaly detection). The detection of ongoing attack in the integrated approach (evidential network) also shows a slight improvement in recall while still maintaining high precision.

Table 4.6: Comparative results for detection tools and integrated solution (EN)

Variable	Metric	Detectors					EN
		ABC	L-ADS	AE	DNN	DBN	
System State = 0	precision	0.052	0.834	0.980	0.731	0.983	**0.989**
	recall	0.993	0.943	0.975	**1.000**	**1.000**	0.985
	f1-score	0.100	0.885	0.978	0.844	0.991	**0.987**
	support	306	131194	131194	131194	131194	131194
System State = 1	precision	**1.000**	0.986	0.994	**1.000**	**1.000**	0.996
	recall	0.663	0.954	0.995	0.910	0.996	**0.997**
	f1-score	0.797	0.970	0.995	0.953	**0.998**	**0.997**
	support	16309	538621	538621	538621	538621	538621

In Table 4.7, the results of the evidential network (EN) with respect to host state variable are listed. The host state is calculated by results of the ABC tool and inferred system state (*normal* or *under attack*). We list the results of these two nodes with respect to CIDDS class label (0 – normal, 1- victim and 2-attacker). ABC tool detects mostly anomalies that refer to attacker hosts, therefore we aggregate class 0 and 1. Analogically, attack/anomaly detection for System State estimation can only distinguish between class 0 (normal) and [1 or 2] (anomaly). As we can see, the ABC tool can detect compromised hosts (attacker) with high precision but quite low recall (0.4). This is expressed in the results of *Host State* estimation. Perhaps using additional context information about host (e.g., log-based anomaly detection) would be beneficial and increase the overall results.

Table 4.7: Results of the Evidential Network for estimation of *Host State* (HS) variable

	ABC Tool		*System State*		*Host State*		
class	0,1	2	0	1,2	HS=0	HS=1	HS=2
precision	0.601	1.000	0.987	0.996	0.498	1.000	0.999
recall	1.000	0.039	0.985	0.997	0.993	0.943	1.000
f1-score	0.751	0.076	0.986	0.997	0.664	0.971	1.000
support	396031	273784	131194	538621	306	5498	10811

In the experiments above, we assume use of binary sensors as input to EN (anomaly/ attack or not). In the future, it is planned to extend the current implementation of evidential network to allow for non-

binary input, e.g. the integration of attack detection algorithms (DNN and DBN) with the output denoting the detected attack type.

4.4.1 Summary and Discussion

The design of the evidential network is a manual exercise based on the analysis of the problem and expert's knowledge. This aspect could be considered as a drawback (since it requires more effort than data-driven approaches) but also a benefit if a representative dataset is not available for tools training and knowledge extraction. The evidential network presented in this section can be used within the complete integration solution, where all detectors are combined. For example, the output from host status could be used in further evidential network node to reason about the attack stage (tactic).
In the presented experiments, we had limited possibilities to demonstrate the accuracy gain using the complementary evidence, mainly due to quite simple dataset and limited types of attack. The accuracy of anomaly detection is already very good and not much could be improved with respect to false positive rate reduction. In future activities, we intend to collect a more representative dataset and perform a more detailed sensitivity analysis to the contradictory evidence reported from the sensors (identify trade-offs). We are planning to perform the evaluation using the pilot site at mnemonic (SOCCRATES first pilot). The anomaly detection algorithms will be tested using adversary emulation tools (e.g. atomic red team[1]), which provide an automated way of executing attacks in the controlled environment. The results of such evaluation will be used for sensor reliability assessment and gap analysis. Based on that, the evidential network structure and configuration will be derived. Finally, we also intend to use a combination of attack steps to showcase the advanced attack detection capabilities of the evidential network.

[1] https://github.com/redcanaryco/atomic-red-team

5 Deployment Plans

One of the objectives of the WP4 is to develop and deploy the AI-based Anomaly Detection (AAD) Component. As mentioned before, this involves combining the detection tools and providing a hierarchical reasoning unit based on their output. To facilitate that, the REASENS framework is used as a potential architecture to integrate detection and reasoning components. Although the framework implements an event-based reasoning (which is usually attributed to SIEM functionalities), it is not intended to replace SIEM. The interface of the AAD with existing SIEM solutions will be implemented at the pilot sites. In the following, we briefly describe the overall structure of the AAD component and discuss potential options for its deployment within the pilot infrastructures.

5.1 Overview

Figure 5.1 depicts an architecture of the AAD component which will be considered in the pilot integration. The AAD component interfaces the infrastructure of the pilot site (mnemonic, Vattenfall) at two points: (1) data sources to collect input for tools (network traffic, logs, etc.) and (2) output of the AAD to existing SIEM solutions. The deployment of the components within the AAD component is supported by the REASENS framework (detection, reasoning, event exchange and normalisation, visualisation).

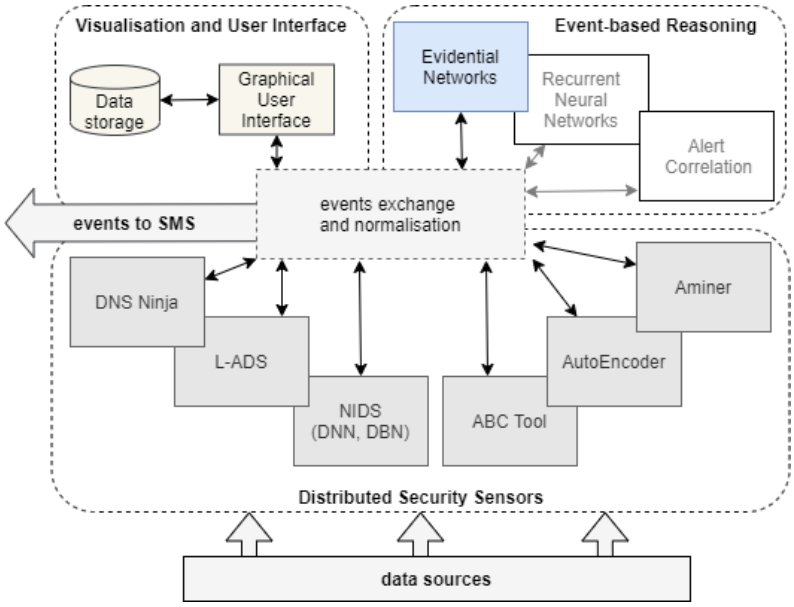

Figure 5.1: AI-based Attack Detection (ADD) Component Architecture.

Our deployment of the AAD at the pilot sites will be done in several stages. Initially, network, DNS and log data will be collected from mnemonic (and later, from Vatenfall) and used to locally train all the detection tools beforehand. It is worthwhile to mention that in the pilot phase, the tools have to be trained at the pilot's premises as data export might not be permitted. This data will consist very likely of only normal host and network activities as, realistically, no major cyber incidents have occurred at the pilot sites and we expect that situation to remain unchanged. This normal data will be used by the tools to create the baseline models, thus incorporating the a-priori knowledge about the systems at the pilot sites. After this step, the AAD will be ready for live deployment, in which new data samples will be labelled/classified and reported in real-time.

In the long run, the detection tools will require re-training. This will be on the one hand necessary for handling changes in the infrastructure (e.g. adding a new server or host to the network), and on the other hand useful to incorporate knowledge about the most recent events (e.g. if a new domain has been found to be malicious, this will be added to the set of existing labels). Periodic re-training of the models could be triggered for instance by the orchestrator, whenever the system is updated, or new assets are being deployed.

Finally, for the concrete assessment of the AAD, we foresee activities related to adversary emulation to collect training and testing data. The AAD will be tested against a set of attack tactics and techniques selected from the MITRE ATT&CK taxonomy corresponding to the four scenarios of Use Case 1 described in Deliverable 2.1 (Eian, 2019). This could be done for instance by using tests from Atomic Red Team or by executing custom tests with a real Red Team. In this manner, it will be possible to assess the detection accuracy of both the individual tools, and the integrated approach with REASENS on a broad range of attack tactics.

The details concerning the data collection at mnemonic and Vatenfall for the initial training of the tools, the re-training process, and the generation of Red Team tests, will be further discussed within WP 7.

5.2 Deployment options

The current implementation of the REASENS framework is based on the publish-subscribe architecture including the MQTT message broker. Our future plans include extending the framework to support integration with the ELK Stack[1] components. Therefore, in our deployment plans, these two options are considered, as depicted in Figure 5.2.

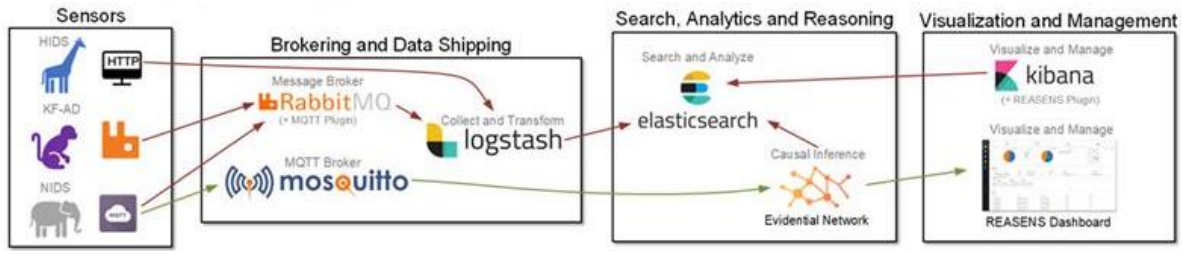

Figure 5.2: Deployment options offered by the REASENS framework.

Detection Tools (Sensors) deployment

We assume that the software components (tools) either already are or are planned to be dockerised[2] to minimise deployment problems. Tools could be deployed individually or together using *docker-compose* (for all components of the AI-based Attack Detection). *Docker-compose* allows to define and deploy an application that is made of several docker images using a YAML file. This produces deterministic deployment behaviour, which should greatly reduce complexity given the number of components involved.

[1] https://www.elastic.co/what-is/elk-stack
[2] https://www.docker.com/

The deployment will include:

- Sensors (e.g., L-ADS, DNS Ninja, ABC Tool, Aminer)
- Reasoning engines (e.g., Evidential Network)
- Software that is necessary for data exchange (e.g., broker Mosquitto, RabbitMQ, Logstash)
- Data translation components for data normalisation
- REASENS server (GUI) + local data storage
- Components that implement external interfaces:
    - Data source adapters
    - Event reporting to the Security Monitoring Solution (SMS)

Brokering and Data Shipping

As mentioned earlier, current REASENS implementation is based on the message broker architecture using publish-subscribe communication between services. That has proved to be very useful for event-driven processing (i.e., processing triggered upon event) in a controlled environment, with sparse events. The sensors need to support MQTT output to publish their alerts/events to MQTT broker. The other services, e.g. event normalisation, evidential network, event logging, can subscribe to the topics of these sensors. The web-based user interface of REASENS allows management of the sensors and subscriptions, e.g., selecting sensors which should be used for reasoning. Finally, to enable output of generated events into the external SMS solution, an Event Logger would need to be implemented, which subscribes to sensor and engines topics, transform the events into required message format and sends them via given interface (syslog, SIEM API).

An alternative solution is to use Logstash from the ELK suite. Logstash allows to define a pipeline to ingest, alter and output data over networks. Technologies supported by Logstash for input/output include RabbitMQ, Kafka, elastic, jdbc, http, syslog and many more[1]. Using Logstash does not imply the use of docker in the deployment. Finally, the same approach (Logstash automatic ingestion/filtering/output) can be used to push events to the SMS solution.

Message exchange and event normalisation

Normalisation component is deployed as a service in the REASENS architecture. In previous projects, it was not required to normalise events, since most of the proposed sensors were custom developed. However, dealing with a wide range of sensors, each generating messages in different formats, it is important to implement normalisation. Logstash provides the benefit of supporting commonly used inputs and can be used to also drop/add “fields” to the alerts.

Visualisation and Management (Data Repository)

Currently, REASENS uses relational database (SQLite) which has proved to be well-suited for the web-based user interface. Nevertheless, dealing with more unstructured data sources, different formats and services, we are inclined to migrate into more flexible databased architecture (nosql). An attractive alternative would be to use Elasticsearch as a data repository. Additional benefits could be achieved while using the other ELK suite components together with Elasticsearch for data management, storage, exchange and normalisation. Then, the dashboard for data visualisation could be included in Kibana as a custom plugin (e.g., for evidential network).

[1] full list of input plugins https://www.elastic.co/guide/en/logstash/current/input-plugins.html

A comparison of the deployment options is summarised in Table 5.1. We can see that both solutions are well suited for deployment of AAD component. The MQTT naturally supports event-driven architectures, easy to extend and is already available in REASENS.

Table 5.1: Deployment Options Summary

	MQTT	ELK Suite
Tools Adaptation	Tool sources are not changed. Tools are executed from the custom extension of MQTT client (required for each tool)	Tool sources are not changed. Logstash for data exchange.
Event Normalisation	Normalisation as service, custom component required	Transform message with logstash (normalisation), plugins available for different formats
Interface with SMS	Implementation of Event Logger component	Logstash configuration
Event-based	Yes	No
Docker	Yes (not required)	Yes (not required)
Management	Yes (available) Monitoring of deployed sensors (events, location, status: active or disconnected) Engine to sensor subscription configuration	Not implemented yet
Graphical User Interface (GUI)	Custom web-based interface with dashboard and management panel	Kibana (+ custom plugin) partially implemented

Nevertheless, there are still several components that need to be adapted for deployment, e.g. adapters for detection tools, event logger (output to SMS) and event normalisation components. On the other hand, ELK suite is becoming increasingly popular and compatible with common, well-established solutions. Providing an interface with ELK stack would be very beneficial, also for further exploitation of project results, e.g. deployment of individual components. What's more, the output from the detectors (sensors) is arbitrary. Availability of different plugins for Logstash could make the normalisation process slightly easier.

As we will get more details about the pilot sites infrastructures, the decision about the best deployment option will be made. We expect that one of the main factors will be how to interface with existing SIEM solutions and ensure compatibility. For instance, some of the new SIEM solutions support ELK (e.g., SIEMonster), whereas other have custom (proprietary) data repository and data collection methods etc. (e.g., ArcSight, Splunk, OSSIM, Qradar). For the latter, communication with syslog messages or using custom API (e.g., Argus API) will be considered. Nevertheless, the final decision about the solutions are planned to be specified in WP7 and will be described in Deliverable D7.1.

5.3 Pilot Integration (Phases)

There are two integration phases planned in SOCCRATES project: the first one aims to perform partial deployment to test the proposed deployment method and identify any potential issues, e.g., with respect to interfaces between tools. Subsequently, the lessons learnt will be used in the second phase, when a full AAD component integration is planned.

In the first pilot phase, a subset of tools will be deployed in the Mnemonic demonstration site. As depicted in Figure 5.3, we are planning to deploy selected detection tools (e.g., DNS Ninja, NIDS) together with the evidential network as a reasoning component.

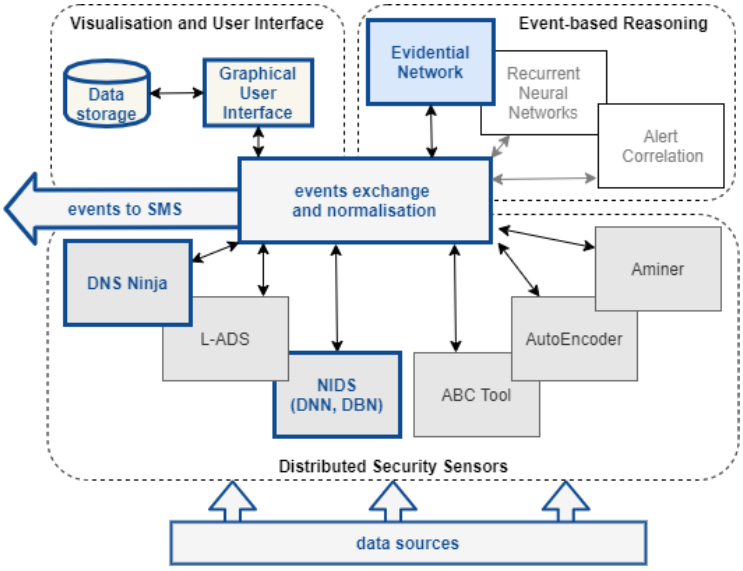

Figure 5.3: Components of ADD planned for deployment in Phase 1 are outlined in blue

In addition, we will provide a graphical user interface to display information about the collected events and response from the evidential network. During the first phase, we assume to gain some insights on the preferred method for event communication (MQTT or ELK) and to provide necessary components to facilitate message exchange and normalisation. Furthermore, it is intended to implement the interface with data sources as well as interface with SMS (mnemonic Argus) in terms of communication between REASENS and Argus via specified API. We also expect more details on consolidation of message format and their further use in SMS. One objective that will be realised at the pilot site is collection of data for detection tools training and testing. This would greatly enhance the evaluation and capabilities of the AAD component.

In the second pilot phase, the experiences from the first phase will be applied to deploy the complete AAD component, including most of the detection tools. At least host and network data analysis should be combined to improve reasoning results. Assuming that the training data is available from the first pilot phase, we are planning to investigate additional event reasoning techniques and objectives.

6 Conclusions

This deliverable provides an overview of the work carried out towards development of the AI-based Attack Detection (AAD) component of the SOCCRATES Platform. An entire suite of intrusion detection tools was described, including existing algorithms developed by the partners (TNO, ATOS, AIT) as well as new deep learning methods implemented within the project (WP4).

Moreover, we presented an approach to integrate these detection algorithms to support situation awareness within the SOCs. Specifically, we investigated the applicability of evidential networks to perform event-based reasoning and infer the current state of the underlying infrastructure, e.g., to detect if the system is under attack or host is compromised.

The numerical experiments performed on the CIDDS dataset showed that integrating multiple detection tools leads to more explainable anomaly events, as well as to an increase in the precision and recall scores. The CIDDS dataset is quite limited in the complexity of attacks, therefore the improvements achieved by tools integration are not very significant. Nevertheless, the accuracy of combined solution is at least as good as the best performing anomaly detection algorithm. This result shows that the evidential network is capable of handling contradictory evidence by accounting for sensors' reliability. Furthermore, it is expected that adding more diverse data sources will improve the reasoning capabilities.

In future activities, we intend to collect a more representative dataset and perform a more detailed sensitivity analysis to the contradictory evidence reported from the sensors (identify trade-offs). Furthermore, we are planning to verify our approach for attack stage detection (tactics) as proposed in Section 3.3.3. To this end, we are planning to leverage existing historical data from mnemonic pilot site (first pilot phase) and employ tools such as atomic red team to emulate adversary and demonstrate the attack detection. The deployment and integration of the tools will be continued in the WP6 and WP7. The final results of task T4.1 (and thus the continuation of work presented in this deliverable) will be integrated with those in T4.2 and presented in the follow-up deliverable D4.3.

7 References

Azami, M. L. (2016). Converting SVDD Scores into Probability. *Computational Intelligence*.

Beukema, W. (2016). *Enhancing Network IntrusionDetection through Host Clustering.* Master's thesis, University of Twente. Retrieved 04 09, 2020, from http://essay.utwente.nl/70560/1/Beukema_MA_EMCS.pdf

Cherifi, W. S. (2017). An Incremental Evidential Conflict Resolution Method for Data stream Fusion In IoT. *Federated Conference on Computer Science and Information Systems*, (pp. 825-834).

Dempster, A. P. (1967). Upper and lower probabilities induced by a mutivalued mapping. *The Annals of Mathematical Statistics*, 325-339.

Eian, M. (2019). *D2.1 Use cases definition & pilot sites requirements.* SOCCRATES.

Friedberg, I. H. (2017). Evidential Network Modeling for Cyber-Physical System State Inference. *IEEE Access, 5*, 17149-17164. doi:10.1109/ACCESS.2017.2718498

Gao, N. G. (2014). An Intrusion Detection Model Based on Deep Belief Networks. *Second International Conference on Advanced Cloud and Big Data*, (pp. 247-252).

Gonzalez-Granadillo, G. D.-Z. (2019). LADS: A Live Anomaly Detection Systems based on Machine Learning Methods. *Conference: International Conference on Security and Cryptography (SECRYPT).* Prague, The Czech Republic.

Haider, W. H. (2017). Generating realistic intrusion detection system dataset based on fuzzy qualitative modeling. *Journal of Network and Computer Applications, 87*, 185-192.

Hawkins, S. H. (2002). Outlier Detection Using Replicator Neural Networks. *Data Warehousing and Knowledge Discovery*, 170-180.

Kent, A. D. (2015). Cybersecurity Data Sources for Dynamic Network Research. *Dynamic Networks in Cybersecurity.* Imperial College Press.

Ou, X. R. (2009). An Empirical Approach to Modeling Uncertainty in Intrusion Analysis. *2009 Annual Computer Security Applications Conference*, (pp. 494-503). Honolulu, HI.

Rezvy, S. P. (2018). Intrusion detection and classifcation with autoencoded deep neural network. *Springer International Publishing* .

Ring, M. W. (2017). Flow-based benchmark data sets for intrusion detection. *Proceedings of the 16th European Conference on Cyber Warfare and Security (ECCWS)* (pp. 361-369). ACPI.

Ring, M. W. (2019). A survey of network-based intrusion detection data sets. *Computers & Security, 86*, 147-167.

Salah, S. M.-F.-V. (2013). A model-based survey of alert correlation techniques. *Comput. Networks*, 1289-1317.

Shafer, G. (1976). *A Mathematical Theory of Evidence.* Princeton, NJ, USA: Princeton Univ. Press.

Smets, P. (1990). The combination of evidence in the transferable belief model. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 447-458.

Tesink, S. (2020). *D2.2 System architecture & Interface Specifications (Initial).* SOCCRATES.

Verhees, L. (2019). *Evaluation of Unsupervised Anomaly Detection on Network Data using Autoencoders.*

Wurzenberger, M. L. (2019). AECID-PG: A Tree-Based Log Parser Generator To Enable Log Analysis. *IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, (pp. 7-12). Arlington, VA, USA.

Wurzenberger, M. S. (2018). AECID: A Self-learning Anomaly Detection Approach Based on Light-weight Log Parser Models. *ICISSP*. doi:10.5220/0006643003860397

Zomlot, L. C. (2011). Prioritizing intrusion analysis using Dempster-Shafer theory. *AISec '11.*

8 Abbreviations

Acronym	Description
ACT	semi-Automated Cyber Threat intelligence
ADG	Attack Defence Graph
AI	Artificial Intelligence
APT	Advance Persistent Threat
BPMN	Business Process Model and Notation
CC	Command and Control
CMDB	Configuration Management Database
CSIRT	Computer Security Incident Response Team
CoA	Course of Action
CTI	Cyber Threat Intelligence
EDR	Endpoint Detection and Response
LDAP	Lightweight Directory Access Protocol
MSSP	Managed Security Service Provider
SDN	Software Defined Network
SIEM	Security information and event management
SOC	Security Operation Centre
SOCCRATES	SOC & CSIRT Response to Attacks & Threats based on attack defence graph Evaluation Systems
TIP	Threat Intelligence platform